FAST FOURIER TRANSFORM GENERATOR

By
A. S. KANADE

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JULY 1975

FAST FOURIER TRANSFORM GENERATOR

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By
A. S. KANADE

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
JULY 1975

EE-1975-M-KAN-FAS

CERTIFICATE

This is to certify that the project work entitled 'Fast Fourier Transform Generator' has been carried out entirely under my supervision and has not been submitted elsewhere for a degree.

R Biswas

R. N. Biswas
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology
Kanpur

ACKNOWLEDGEMENT

I wish to convey my deep sense of gratitude to Dr. R.N. Biswas for his continuous guidance and encouragement through out the course of this work. I am particularly indebted to him for suggesting some novel ideas without which it would not have been possible to continue with this work.

My sincere thanks are due to the Staff of the Electrical Engineering Department who helped towards the completion of this project. I also wish to thank Mr. K.N. Tewari for his invaluable assistance in typing this report.

A.S. Kanade.

CONTENTS

LIST OF FIGURES

ABSTRACT

An Arithmetic unit capable of performing the butterfly computations occurring in a FFT algorithm has been designed and fabricated. This AU is a separate self contained module and can be used to compute radix '2' FFT of upto 2K number of samples. The conventional ROM containing trigonometric coefficients has been replaced by a much smaller ROM containing ten incremental coefficients. With the help of these, the AU generates the required coefficients. The AU must be interconnected with the main control unit and a random access memory module to form a complete Fast Fourier Transform Generator (FFTG) system.

INTRODUCTION

Major advances made by engineers in producing faster data processing systems usually stem from the development of Electronic devices. But this is not the case with signal and data analysis applications involving Fourier transformation. Here the advances have bee n triggered not by electronics but by mathematics.

Fourier transformation is a useful tool in extracting the information contained in many kinds of waveforms such as seismic waves, electronic encaphalograms and data signals telemetered from deep space. Many approaches have been taken to find the energy content of frequencies. One familiar and inexpensive method calls for a bank of filters. But this is an analog approach which is inherently limited in resolution and flexibility. These disadvantages are overcome by taking Discrete Fourier Transform (DFT) of a record of samples of a continuous waveform. The integral is replaced by finite summation

$$F\{f\} = DFT(x(n)) = \sum_{0}^{N-1} x(n) \exp[-j \frac{2\pi}{N} f.n]$$

where x(o), x(1), ..., x(N-1) are the samples of the continuous waveform x(t). The above expression can be rewritten as

$$F\{k\} = \sum_{n=o}^{N-1} x(n) W^{nk}, \; k = 0,1,\dots,N-1$$

and $$W = \exp[-\frac{2\pi i}{N}]$$

In the straightforward brute force technique, all the Fourier coefficients have to be calculated separately. In the case of a real function, only N/2 coefficients have to be calculated since those corresponding to more than half the sampling frequency are complex conjugate of those below. Thus a total of $2N.N/2 = N^2$ real multiplications and additions are required to compute the coefficients. This effort becomes phenomenal for any useful number of samples. In addition, 2N memory locations are required to store the data and the coefficients.

In 1965, J.W.Cooley and J.W.Tukey of Bell laboratories developed an algorithm which achieves spectacular computational savings over the brute force technique. Called the 'Fast Fourier Transform' (FFT), this algorithm has made possible the use of Fourier transformation techniques in many fields where it was previously too expensive because of the large numbers of computations involved. The algorithm is based on the fact that if the number of samples 'N' is divisible by an integer 'r' (1,2,3 ...), then the Fourier transformation problem can be simplified by obtaining the transforms of 'r' groups of N/r points and combining the results. The total number of calculations is minimised by repeating this factoring process in an appropriate way. The computational saving is of the order of $N/\log_2 N$.

The DFT is a very useful analytical tool which can be used for various data processing applications. It can be used by analytical chemists to measure NMR spectra. Structural designers can use it to determine the transfer function and vibration modes of a structure. Brain researchers can use it to measure spectra of brain waves. It can also be used by behavioural scientists, psychophysicists, biomedical researchers, process control system designers, oceonographers and geophysicists. Most of these diversified uses have become possible only because of the FFT.

The details of Cooley-Tukey algorithm are described in the first chapter. Second chapter contains various aspects of system design, design alternatives and the choice of final design. The arithmetic unit, its hardware and the operation is described in third chapter. A tentative design for the main control unit is presented in the fourth chapter.

Chapter 1

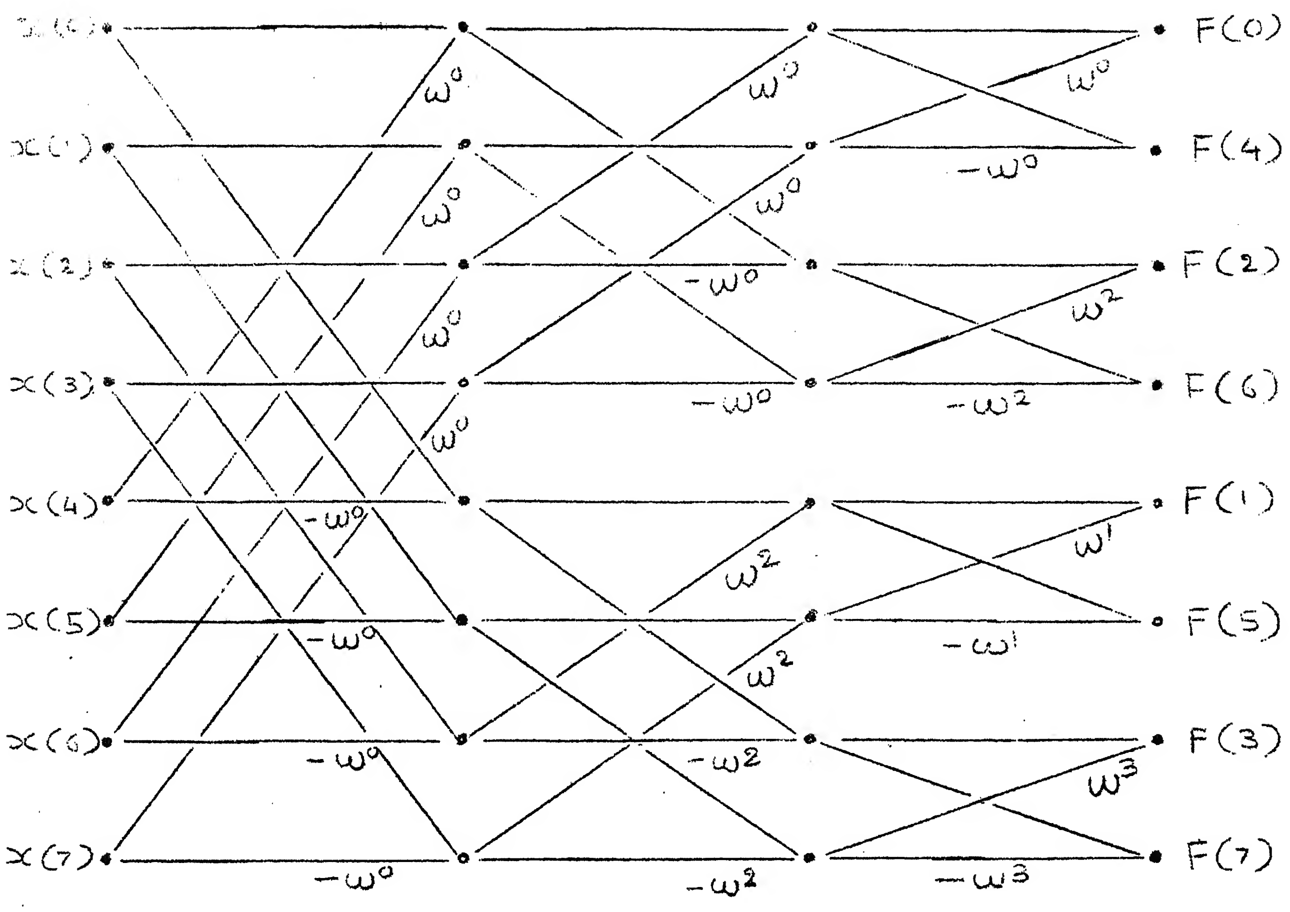
MATHEMATICAL BACKGROUND OF FFT

As described previously, the Cooley-Tukey algorithm is based on the property of factoring the record of samples, taking the DFT of the factors and then combining the results. If the number of samples $N = r^m$ where 'r' is an integer, then the factoring can be continued till only 'r' point transforms have to be calculated in the first step. The algorithm is then called radix 'r' FFT. From hardware realization point of view, only radix '2' FFT is important which is described in the following section.

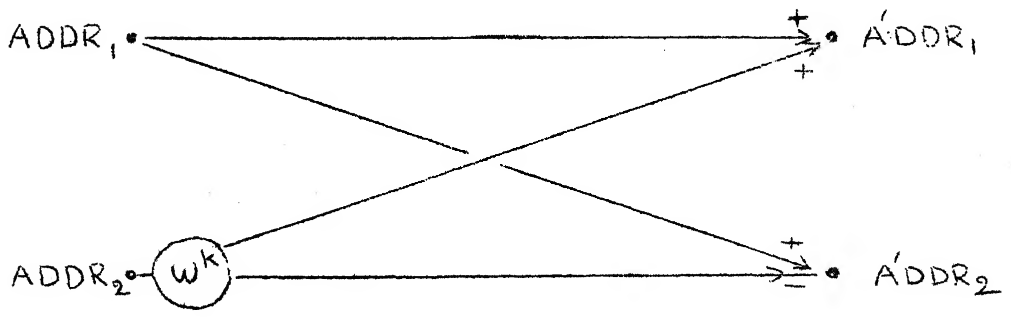
1.1 <u>Radix '2' FFT</u>

If the number of samples $N = 2^n$, then the factoring can be continued till there are N/2 groups of 2 points. These two-point transforms can be calculated easily. The two-point transforms are then combined to produce four-point transforms, four-point transforms are combined to produce eight-point transforms and so on. If this process is continued, then after $\log_2 N$ such steps, the complete 'N'-point transform is obtained.

Thus it is observed that for an 'N'-point transform, there are $\log_2 N$ stages of combination. These are called iterations. In each iteration there are N/2 basic computations. The sequence of operation is illustrated by the signal flow graph on the next page (Figure 1.1).



SIGNAL FLOW GRAPH FOR '8' POINT FFT



BASIC 'BUTTERFLY' COMPUTATION

Figure 1.1

The entire transform is obtained by repeating a basic computational pattern (colloquially called the butterfly). The flow graph is constructed for an eight-point transform; however the symmetry of the graph suggests how it can be expanded for any number of points.

1.2 The Basic Computation

The basic computation of FFT is the generation of the two-point transform. It consists of accessing two complex numbers from two memory locations, multiplying the second number by an appropriate trigonometric weight W^k, adding it to the first number and storing back in the first location; again multiplying the second number by a weight $W^{(k+N/2)}$, adding it to the first number and storing back in the second location. 'W' is defined to be $\exp(-2\pi j/N)$.

We observe that $W^{(k+N/2)} = W^k.W^{N/2} = -W^k$ since $W^{N/2} = \exp(-\pi j) = -1.0$. Hence we need to perform only one complex multiplication instead of two. Then the basic computation is modified as follows.

Two complex numbers are accessed from two memory locations. The second number is multiplied by W^k. The result is added to first number and stored in the first location, subtracted from the first number and stored in the second location.

Thus each computation involves one complex multiplication or four real multiplications. The total number of real multiplications in the transform becomes $2N.\log_2 N$ instead of N^2 in the brute force technique. The saving in computational effort is apparent. For a 4-K point transform, the saving is of the order of 170.

1.3 The Trigonometric Weights

For an 'N' point transform, 'N' complex weights are required. These are nothing but 'N' roots of unity. As explained in the previous section, only N/2 are independent, the remaining N/2 are just negative of these independent weights. Furthermore only N/4 independent numerical values make up these N/2 weights. This is because of the fact that $\sin(90-\theta) = \cos\theta$.

For the computation of the transform these weights are required in reversed bit order, and not in their natural order. Furthermore, not all weights are required in all the iterations. Only one weight is required in the first iteration, two are required in the second iteration, four in the third and so on. All N/2 weights are required in the last iteration.

The reversed bit ordering of weights implies that a weight W^k is required not in its natural order, i.e. at kth place, but is required at the $\underline{k}$th place where '$\underline{k}$' is the reversed bit integer of 'k'. A reversed bit integer is obtained by reversing the order of bits in the binary form. The reversed bit ordering is an essential feature of the radix '2' FFT algorithm and is not confined only to the ordering of weights, as we shall observe in the next article.

1.4 Reversed Bit Ordering of Data and Results

Another property peculiar to the FFT is that, for a naturally ordered sequence of samples, the Fourier coefficients do not occur in the natural sequence but in the bit reversed sequence. To access a particular coefficient, we must find the reversed bit integer of the address and access the coefficient from that location. For example, in the eight-point FFT, coefficient F(3) occurs in location 6, because 6(110) is the reversed bit integer of 3(011).

The above process is called 'decimation in frequency' since the Fourier coefficients occur in the scrambled form. On the other hand, if Fourier coefficients are to be obtained in the natural sequence, then the data or the record of samples must be scrambled in the reversed bit order. This is called 'decimation in time'.

1.5 Choice of'N'

Since the time function to be transformed must be sampled at discrete intervals, say Δt, only a finite number 'N' of such samples can be taken and stored. The record length T is then $T = N.\Delta t$. The effect of finite Δt is to limit the maximum frequency that may be sampled without aliasing error to $f_{max} = 1/2\Delta t$. Any components above this Nyquist frequency are folded back onto frequencies below f_{max}. In practical measurement situations, this aliasing presents little or no difficulty since f_{max} can be chosen to include all significant components of the input signal or a low-pass filter may be used before the sampler to eliminate any strong components above f_{max}. The spectral resolution is given by $\Delta f = 1/T$ i.e. to get fine resolution, record length must be large enough.

In general, f_{max} is limited to a few KHz in many practical systems. For example, NMR spectrometry requires a range of 2000 Hz, with a resolution of 1 Hz. This implies that sampling rate must be atleast 4000 Hz and the record length 1 second, so that N = 4000. N is chosen to be 4096 or 4-K in this case for the FFT to be applicable.

1.6 Address Generation

Each butterfly computation requires a pair of addresses. These addresses follow a repetitive logical pattern. The sequence of addresses is different in each iteration. We can determine the logic for 8-point FFT and extrapolate the results for an 'N' point FFT.

In the example, the address pairs in the first iteration are (000,100), (001,101), (010,110) and (011, 111). It is observed that second address in a pair is obtained by inverting the first bit of the first address. The address pairs are (000,010), (001,011), (100,110) and (101,111) in the second iteration. Here the second address is obtained by inverting the second bit of the first address. In the third and last iteration, address pairs are (000,001), (010,011), (100,101) and (110,111). Here the third bit of the first address is inverted to obtain the second address. The logical pattern suggests that in the k^{th} iteration, the second address in a pair can be obtained by inverting the k^{th} bit of the first address.

Again observing the address pairs of the example, we find that the first bit of the first address is always zero in the first iteration, second is zero in the second iteration and third is zero in the third iteration. Thus the first addresses are (000,001,010 and 011) in the first iteration,(000,001,100 and 101) in the second iteration and (000,010,100 and 110) in the third iteration. We observe that the sequence is strictly natural binary if the permanent zero bit is suppressed.

Chapter 2

SYSTEM DESIGN CONSIDERATIONS

Since the Fast Fourier Transform Generator (FFTG) is a special purpose hardwired calculator, its organization is essentially the same as a computer. It must have a memory for storing data and results, a processing unit which carries out the calculations and a main control unit which controls the processor and the memory. There are two main alternatives for the processing unit which are given below.

2.1 Sequential Versus Cascade Organization

The task of the processing unit is to carry out the butterfly computations. In the sequential organization, the processing unit has one Arithmetic Unit (AU) which is hardwired to carry out a basic computation. It carries them out one by one and is the slowest and cheapest of all organizations. In cascade organization, the processing unit has $\log_2 N$ arithmetic units which start computing all the iterations more or less simultaneously. This organization is costly and is justified only if a real time analysis must be done or a large amount of data must be processed at a very high speed. The sequential processor is much cheaper and ideal for off line requirements. Since our aim is not processing of real time data, the sequential organization is chosen.

2.2 Record Length of Samples

As calculated in Section 1.5, 4000 point FFT is sufficient in most of the practical situations. To meet the radix '2' requirement, we choose maximum N to be 4096 or 4-K points. It is also desirable to be able to generate FFT of lesser number of points without changing the hardware. This brings us to the concept of modularity which is dealt with later in more detail in Chapter 4.

2.3 Word Length

There are two main forms in which the FFTG can output the results. One is to plot the Fourier coefficients on an x-y recorder or display them on an oscilloscope. Other is to print them out in decimal form on a typewriter. The second form is necessary where results must be accurately known for further analysis. The direct display or plotting does not require more than 8 bits/word accuracy. For the second alternative, number of bits must be more than 12/word to take care of the computational errors. Hence the tentative word length must be 12-16 bits/word. Greater word length costs more in terms of hardware. For very good accuracy, it is better to use a general purpose computer system.

2.4 Alternatives for Weight Generation

As mentioned earlier, an 'N' point FFT requires N/2 trigonometric weights. These constitute N/4 independent numerical values. The first alternative is to have a Read Only Memory which stores these N/4 independent values. For 4-K point FFT, we require a ROM with the capacity of storing 1-K words, 12-16 bits/word. Such ROM's are not available as off the shelf items and getting them custom made is costly, unless required in large quantities.

The second alternative is to generate these trigonometric values in binary form as a multiple output switching function whose input is the address of the weight. This synthesis is almost impossible without the availability of efficient computer programs for minimization of truth tables. Even with their help, the necessary number of gates for realization may turn out to be very large.

The third alternative is to compute the required complex weight. This possibility is especially attractive as the AU already has the capability of generating new complex numbers by complex multiplications. In the next section, we look into this alternative in more detail.

2.5 Complex Weight Generation

As pointed out in Section 1.3, the weights are always required in reversed bit order. This fixed ordering indicates that as we know the next weight required, it can be generated by multiplication of the previous weight and an appropriate complex number. This complex number is nothing but 'W' raised to the power of the difference between the powers of the previous weight and the new weight.

A computer program was run to arrange the weights in reversed bit order and calculate the increments. It turned out that 11 independent weights were required for 4-K point FFT, 10 for 2-K point FFT and 9 for 1-K point FFT. Of these, one is trivial, its value being $\exp(-\frac{2\pi j}{N}\frac{N}{4}) = -j$. Multiplication by this just implies interchanging the real and imaginary parts of the previous weight and inverting the sign of the new real part. Thus the number of nontrivial weight increments turns out to be $\log_2 N-2$ for 'N' point FFT which is 10 for 4-K point FFT and 9 for 2-K point FFT. This number is small enough for the ROM to be realized using diode matrices. All the other weights can be realized by multiplying the previous weight and one of these chosen according to a fixed pattern. This pattern is described in detail in Section 3.1 and 4.5.

2.6 System Description

Memory: For a 4-K point FFT, 4-K locations for storing real and 4-K for storing imaginary parts of the Fourier coefficients are necessary. Since the memory module is the most costly of all, making it dedicated for the use of FFTG is uneconomical. Hence it was decided to fabricate a general purpose random access memory with single word read/write capability. The memory is a 4-K word, 20 bits/word memory. Hence it can be used for calculating a 2-K point FFT.

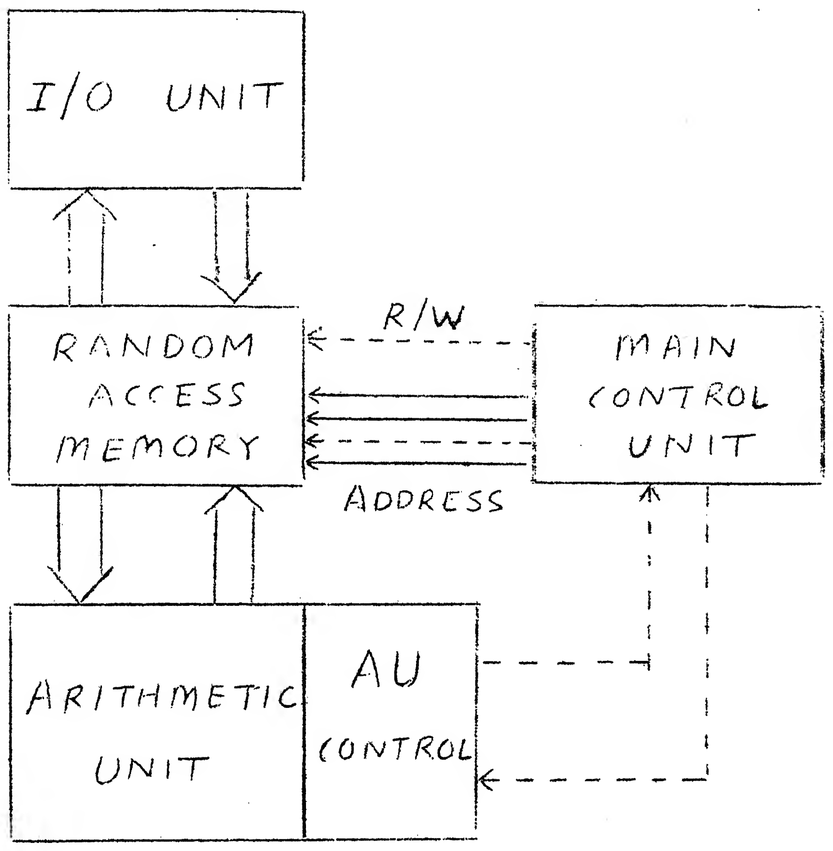
Main Control: Main control generates addresses for the butterfly computations and sends other control commands to AU and memory. The logic for address generation has already been dealt with in Section 1.6. The other details are given in Chapter 4.

Arithmetic Unit: Design aspects of AU are given in details in the next section.

It was decided to fabricate all three subsystems separately and interconnect them later whenever desired. Two of the subsystems memory and main control can find use as parts of other systems as well. The system block diagram is given on the next page (Figure 2.1).

2.7 Design Considerations for AU

The AU has to perform two complex multiplications, two additions and two subtractions. In terms of basic computations, this involves eight real multiplications, four additions and four subtractions. These operations can be performed in bit serial or bit parallel form. Though bit parallel operations are very much faster, they are much more costly in terms of increased hardware. Hence the bit serial form of addition and subtraction was chosen.



⟹ Denotes Data Paths

- - - -> Denotes Control Signals

SYSTEM BLOCK DIAGRAM FOR

FFTG

Figure 2.1

2.8 Single Multiplier Versus Four Multipliers

Each complex multiplication requires four real multiplications. These can be done sequentially by one multiplier or together by four multipliers. The single multiplier scheme requires lesser processor hardware. On the other hand, the AU control becomes more complex because it has to control eight multiplications and associated real and imaginary additions and subtractions which take place sequentially.

The four multiplier scheme requires more processor hardware but is faster than the single multiplier scheme. Moreover, the AU control is much simpler because it can control all the operations in parallel with the help of same control commands. It turns out that the total hardware including both processor and control, is about 25 percent more in the case of the four multiplier scheme. However, its execution time is about one third of that obtainable with the single multiplier scheme, which is a substantial speed gain for a small additional amount of hardware. The four multiplier scheme was chosen for the AU.

Chapter 3

ARITHMETIC UNIT

The Arithmetic unit of the FFTG performs one complex multiplication, two additions and two subtractions involved in the butterfly computation. It also performs an additional complex multiplication when required to generate the new weight. The basic design decisions for the AU were given in the last chapter. In this chapter we examine the AU hardware in detail. The AU block diagram is given on the next page (Figure 3.1).

The AU hardware consists of four multipliers, one adder, one subtractor, two data registers,one ROM buffer register,one weight register, two buffer registers, six serial two's complementers, some flip-flops to store the sign bits of data and intermediate results and some additional gating and combinatorial logic. It also has a Read only Memory and decoder which are described below.

3.1 ROM for Weight Increments

As was mentioned in the previous chapter, there are 10 independent nontrivial increments which must be preserved in a ROM. These 10 increments are for 4-K point FFT, but the same increments hold good for any other 'N' point FFT for 'N' less than 4-K. Each increment is a complex number with a real and an imaginary part. Each number has 14 magnitude bits and

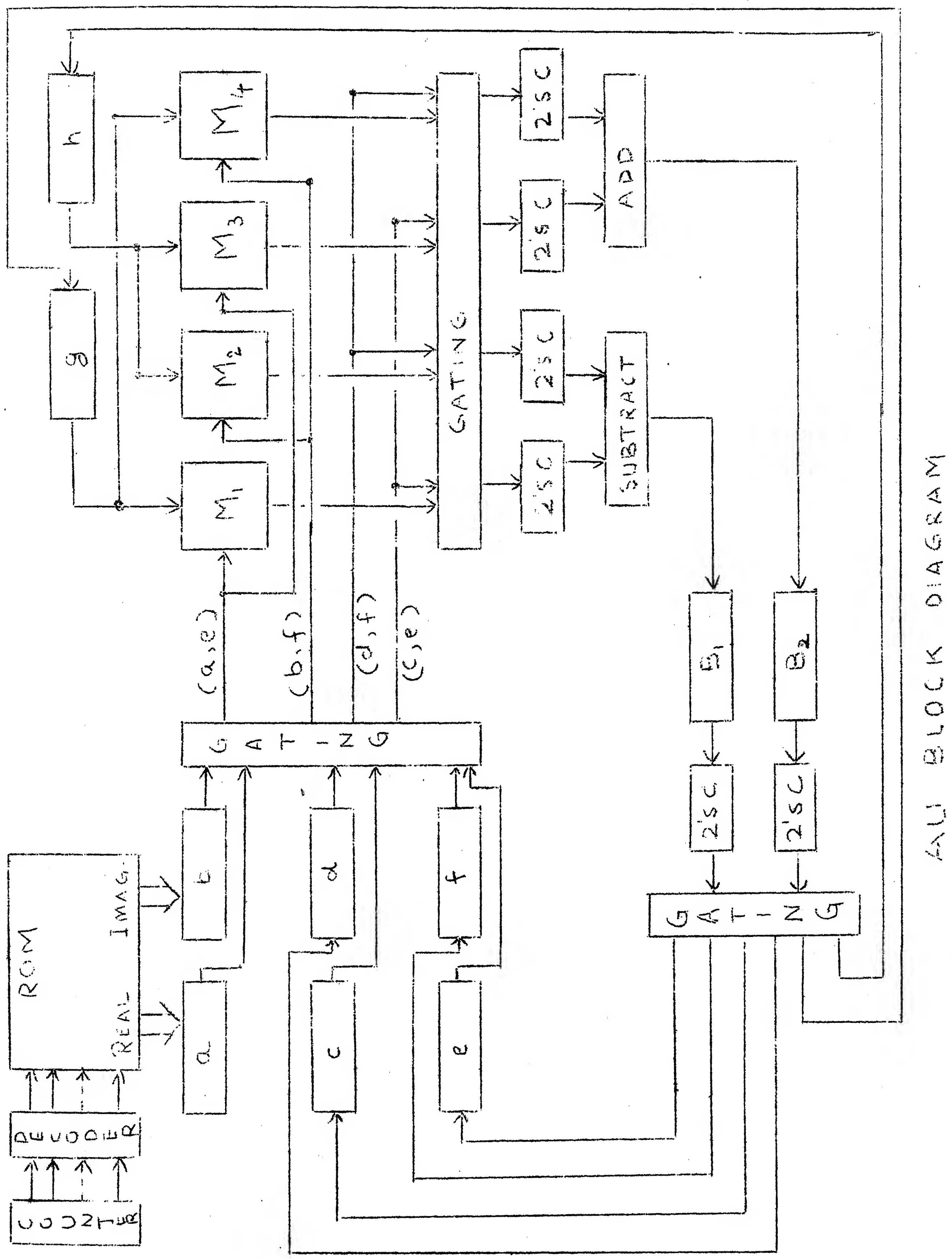

ALU BLOCK DIAGRAM
ROM
REAL
IMAG.
COUNTER
DECODER
a
b
c
d
e
f
g
h
GATING
(a,e)
(b,f)
(d,f)
(c,e)
M_1
M_2
M_3
M_4
GATING
2'sC
2'sC
2'sC
2'sC
ADD
SUBTRACT
B_1
B_2
2'sC
2'sC
GATING

Figure 3.1

one sign bit.. The choice of 14 as number of magnitude bits is explained in the next section. Thus a ROM having 20 x 15 organization is indicated. Each of the ten lines selects two 15-bit words. To select one of these increments, a counter decoder is used. To generate the logic for this decoder, a computer program was run to arrange the increments in the order they are required. By looking at the time of occurance of the increments, a simple logical pattern emerges which consists of detecting particular bit patterns of a 10 bit counter. There are ten such patterns for the ten coefficients. Each is realized with the help of multi-input NAND gates. Incrementing the counter itself requires another logical pattern and is under the control of main control unit.

3.2 Word Size for the AU

It is convenient to use 16 bit registers in the AU. If one bit is to be left for overflow which will always result if data is normalized to 1, then this leaves us with 15 bits. Of these 14 bits are magnitude bits and one bit in the sign bit.

The overflow results because of the repeated additions and subtractions of intermediate results. This also causes the Fourier coefficients to be computed within a scale factor of 'N' which will require longer registers. To scale down the coefficients, they must be divided by 'N'. One simple way to do this is to shift the result one bit to the right in every iteration. This amounts to division by $2^{\log_2 N} = N$.

3.3 Interpretation of numbers

The numbers occuring in FFT can be interpreted either as fractions or integers. If they are interpreted as integers, the binary multipliers have to preserve the full product, necessitating more hardware. On the other hand, interpretation as fractions leaves us free to reject the lower order bits of the product and still retain the same interpretation. Hence for all further discussions we assume the data to be normalized to 1, with result also being normalized to 1 after one right shift.

3.4 The Complex Multiplier

This consists of four binary multipliers. Each multiplier is identical and multiplies two 16 bit numbers. The product is 16 bits long since the least significant 16 bits are rejected. The multipliers use the add and shift algorithm. Each multiplier has two 16 bit registers, a 16 bit adder and 16 AND gates. Each multiplier forms one of the four products needed to generate a complex product. The outputs of the multipliers are hardwired to the inputs of the adder and subtractor. The output of the subtractor forms the real part and the output of the adder forms the imaginary part of the complex product.

3.5 The Adder and Subtractor

The inputs to the adder and subtractor are from the multipliers and the data registers. Either can be selected with the help of gating. The same adder and subtractor is used in complex multiplication as well as the addition and subtraction involved in the butterfly computations.

The input numbers are two's complemented before being added or subtracted. The results are converted back to sign magnitude form before storing back in data registers. Thus there are four two's complementers at the input and two at the output. These are standard serial two's complementers. The carry flip-flops in the adder and subtractor are used to store the sign bit of the result which occurs as the most significant bit of the result. These sign bits control the operation of output two's complementers.

3.6 The Data and Coefficient Registers

The AU has a number of registers to store data, coefficients and intermediate results. There are two data registers which hold the two complex numbers read from the RAM. Each consists of two 16-bit registers. These are parallel input, parallel output, serial input, serial output registers. ROM buffer register holds the complex weight increment read from the ROM. This also consists of two parallel input, serial output, 16-bit registers. The weight or coefficient register which holds the current trigonometric weight consists of two serial input serial output 16-bit registers. These must, in addition, have parallel input capability to load W^0 or 1-j0 at the start of each iteration. All of the above registers are fabricated using 7495 universal register IC's.

In addition, the AU has two 16-bit serial in, serial out buffer registers to store the result of the adder and subtractor. This temporary storage is necessary because the sign bit of the result appears in the end and the result must be two's complemented before being stored in data registers.

Some flip-flop flags are used to store the sign bits of the numbers being processed. The sign bits of the multiplicand and multiplier are exclusive OR'd to generate sign bit of product. A small amount of AND-OR-logic is used to select the numbers to be multiplied and/or the numbers to be added and subtracted.

3.7 'One of Four' Selector

Since the RAM is single word read write memory, the computed results have to be stored one by one. There are four real numbers to be stored hence the main control unit must select them one by one and write them in the memory. 'One of four' selector is suitable for this purpose and can be easily fabricated using 2 input and 4 input NAND gates. While reading from the memory, 'one of four' selector is not required since all the register inputs can be connected to the memory output simultaneously. Data is selectively written in one of the registers by giving a load pulse to that particular register.

3.8 AU Control

AU control generates the necessary clock pulses and gating commands necessary for the operation of the AU. Its working is governed by a control algorithm which has twelve steps, sixteen clock pulses per step. Thus execution of the control algorithm takes a maximum of 192 clock pulses. The control algorithm is given below.

Notations:

Data_1 and Data_2 are the two words fetched from the memory.

$$\text{Data}_1 = c + jd, \quad \text{Data}_2 = e + jf$$

The ROM register stores the increment a+jb. The weight register stores the previous weight g+jh. The multipliers are M_1, M_2, M_3 and M_4. 'I' is the interchange command. B_1, B_2 are the buffer registers. C(x) denotes contents of register and $\leftarrow$ denotes loading operation.

Addition and subtraction are in two's complement.

```
Begin
Step 1:   C(a) loaded in $M_1$ and $M_3$
          C(b) Loaded in $M_2$ and $M_4$
Step 2:   C(g) multiplies C($M_1$) and C($M_4$)
          C(h) multiplies C($M_2$)and C($M_3$)
          C(g) and C(h) rotated if I=0
          C(g) and C(h) interchanged if I=1
Step 3:   C($B_1$)$\leftarrow$C(a)·C(g) - C(b)·C(h)
          C($B_2$)$\leftarrow$C(a)·C(h) + C(b)·C(g)
Step 4:   C(g)$\leftarrow$C($B_1$)
          C(h)$\leftarrow$C($B_2$)
Step 5:   C(e) loaded in $M_1$ and $M_3$
          C(f) loaded in $M_2$ and $M_4$
Step 6:   C(g) multiplies C($M_1$) and C($M_4$)
          C(h) multiplies C($M_2$ and C($M_3$)
          C(g) and C(h) rotated.
Step 7:   C($B_1$)$\leftarrow$C(e)·C(g) - C(f)·C(h)
          C($B_2$)$\leftarrow$C(e)·C(h) + C(f)·C(g)
Step 8:   C(e)$\leftarrow$C($B_1$)
          C(f)$\leftarrow$C($B_2$)
Step 9:   C($B_2$)$\leftarrow$C(C) + C(e)
          C($B_1$)$\leftarrow$C(C) - C(e)
Step 10:  C(C)$\leftarrow$C($B_2$)
          C(e)$\leftarrow$C($B_1$)
Step 11:  C($B_2$)$\leftarrow$ C(d) + C(f)
          C($B_1$)$\leftarrow$ C(d) - C(f)
```

Step 12: $C(d) \leftarrow C(B_2)$

$C(f) \leftarrow C(B_1)$

End

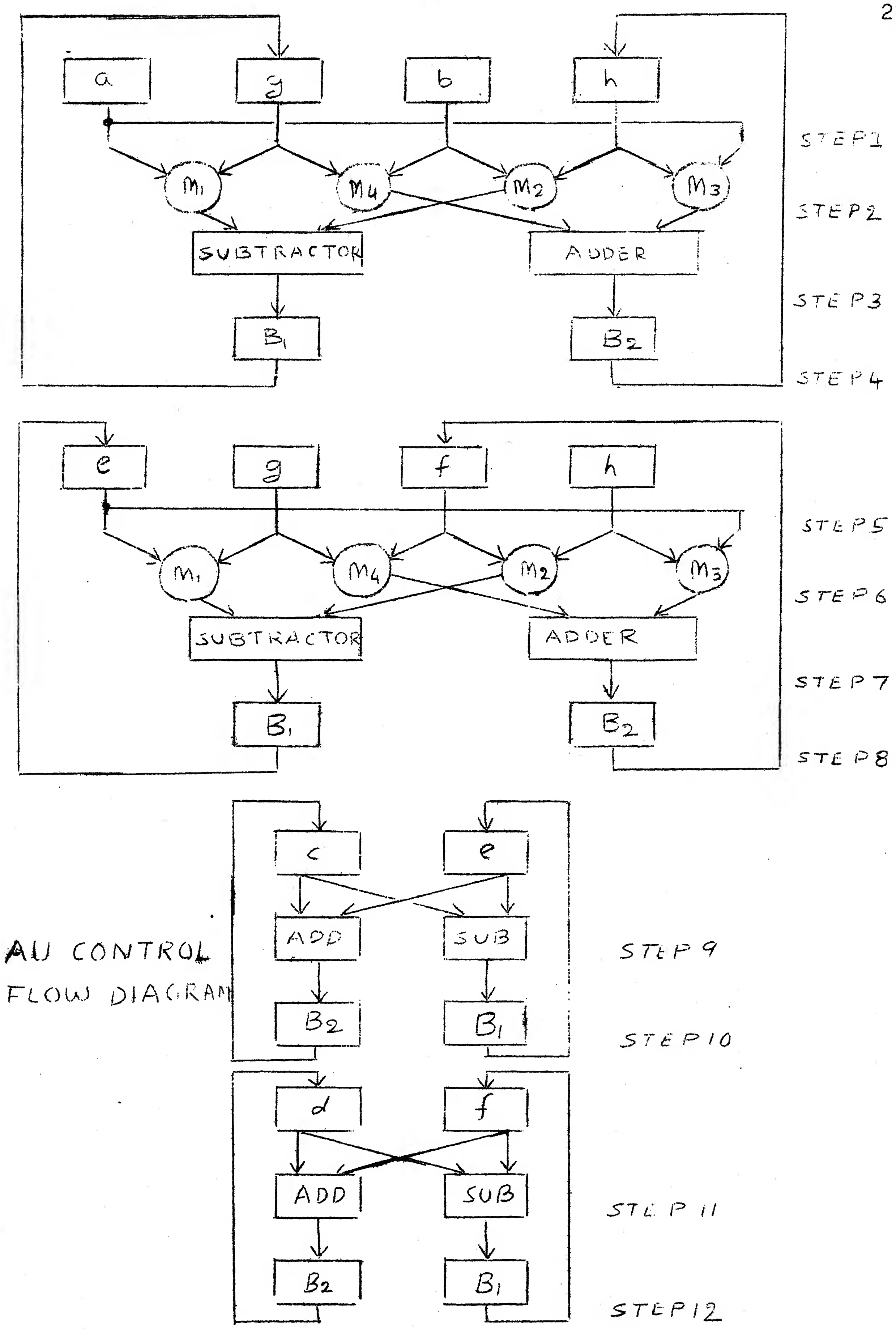
The pictorial representation of the control algorithm is given on the next page (Figure 3.2).

As is observed in the control algorithm, each complex multiplication takes place in four steps. Multiplicand is loaded in the MP register of the multiplier in the first step. Multiplication by summation of partial products takes place in the second step. The addition and subtraction of real products to form real and imaginary parts of the complex product is performed in the third step. In the fourth step, the complex number generated is loaded back in the coefficient register or the data register, as the case may be.

The first four steps in the control algorithm are necessary only for the generation of the new weight. This suggests two alternatives for the synthesis of AU control

3.9 Alternatives for AU Control

The first alternative is to make two separate controls. First control operates when the new weight has to be generated and all the twelve steps must be executed. Second control operates when only the latter eight steps are to be executed. This approach



a
g
b
h
m1
M4
M2
M3
SUBTRACTOR
ADDER
B1
B2
STEP1
STEP2
STEP3
STEP4
e
g
f
h
STEP5
STEP6
STEP7
STEP8
c
e
ADD
SUB
STEP 9
STEP10
d
f
STEP 11
STEP12
AU CONTROL FLOW DIAGRAM

Figure 3.2

reduces the average execution time of the AU but is quite costly in terms of requirement of hardware.

A better way is to make a control which always executes twelve steps. The clock is suppressed when the first four steps are to be skipped and starts only at the beginning of the fifth step. This permits a simple counter-decoder type of synthesis for the AU control.

3.10 Counter-decoder Synthesis

The cycle counter is a 12x16 state eight bit synchronous counter. A trigger pulse sets a latch which allows the clock to go through an AND gate. The same clock advances the counter. After 192 pulses have passed through, the latch is reset and the clock stops. A decoder detects the state of 'R' or 'I' commands (details in Chapter 5) and allows either 128 or 192 pulses to pass through. Another decoder generates gating commands according to the state of the counter. A second decoder generates clock pulses for various sequential circuits by opening or closing AND gates.

In addition to AU control, the main control also has to give various commands to the AU. The details are given in the next chapter.

Chapter 4

MAIN CONTROL UNIT

To execute the FFT, a particular indexing pattern is required which is different for each of the $\log_2 N$ iterations. The logic for the address pair generation has already been described in Section 1.6. Since the pattern is repetitive and simple, it can be easily hardwired. For the same reason, modularity can be easily introduced in the control unit so that the same unit works for any number of points in the computation of radix-2 FFT.

The main function of the control unit is to keep track of the iteration number and accordingly generate a pair of addresses. If decimation in frequency is used, it is observed that the first iteration involves one group of butterflies, second involves two groups, third involves four groups and so on. The 8 point FFT example illustrates this clearly.

4.1 Iteration Register

There are two ways of keeping track of the iteration currently in progress. The first one is to have a straight binary counter which has $\log_2 N$ states and hence $\log_2(\log_2 N)$ bits. Since the iteration number controls other sequential circuits to produce addresses, the output of the binary counter has to go through substantial amount of decoding. The second alternative is much more elegant and easier to implement. Instead of a straight binary counter, a ring counter or shift register is needed. Initially all the bits of this register which will be called 'Iteration Register', abbreviated IR, from now on, are cleared to zero. When the computation starts, the first or the left most bit is made 1. Whenever an iteration ends, it shifts this single 'one' one place to the right. The left bits are cleared. Thus only the second bit is '1' in the second iteration, and the third bit is '1' in the third iteration. In general, only k^{th} bit is '1' in the kth iteration, rest of bits are zero. When this single '1' is shifted out of the last or the extreme right bit of the IR, it signifies that all the $\log_2 N$ iterations are over and the process stops. The $\log_2 N$ outputs of the IR control the address counter in a very simple way which we shall see in the next section.

4.2 Address Counter

As described in Section 1.6, the second address in a pair in the k^{th} iteration is obtained by inverting the k^{th} bit of the first address in the pair. Since we already have an iteration register having only k^{th} bit equal to '1' in the k^{th} iteration, the individual bits of the second address can be obtained simply by means of a set of $\log_2 N$ 2-input OR gates, having the corresponding bits of the IR and the address counter as their inputs. Thus the address counter has to generate only the first address in a pair.

As we observed in Section 1.6, the k^{th} bit of the first address in the k^{th} iteration is always zero. The first addresses are in a natural binary sequence if this zero bit is suppressed. This suggests that the address counter can be synthesized in a very simple way. The counter has $\log_2 N$ bits and hence the same number of flip-flops. Instead of connecting the output of each flip-flop only to the clock input of the next flip-flop as in a binary ripple counter, it is also connected to the clock input of the flip-flop after the next one in the line through an AND gate, as shown in Figure 4.1. The other input of the AND gate is the corresponding bit of the IR. For example, the input of the AND gate corresponding to the k^{th} most significant bit flip-flop is the k^{th} most significant bit of the IR. In addition the inverse of the corresponding IR bit is connected to the clear terminal of the corresponding flip-flop. Thus if the k^{th} bit of the IR is '1' then the kth flip-flop is cleared for the complete duration of the k^{th} iteration, thus making the k^{th} bit of the first address permanently zero. It also causes the corresponding AND gate to connect the output of k-1th flip-flop to the clock input of the k+1th flip-flop thus skipping the kth flip-flop entirely. This counter counts in the desired sequence and generates the first address in a pair. The second address is generated combinatorially and hence both addresses in a pair are available simultaneously.

4.3 One of Two Selector

Since the RAM is only single word read/write, each address has to fetch the real part and then the imaginary part. Thus a one of two selector is required for the two addresses. The first address is connected to the memory address input, the real part read and loaded in the corresponding register of the AU. Then imaginary part in read and loaded in the second register. Then the second address is connected and two accesses and loading done in a similar manner. Thus a total of four accesses per computation are required. After the AU has signalled end of computation, the real and imaginary parts of both data words are stored in the memory one by one in a similar way using the one of two selector.

4.4 Computation Counter

A computation counter with N/2 states or $\log_2 N-1$ bits is required to keep track of the computation currently being executed. This is a simple ripple carry binary

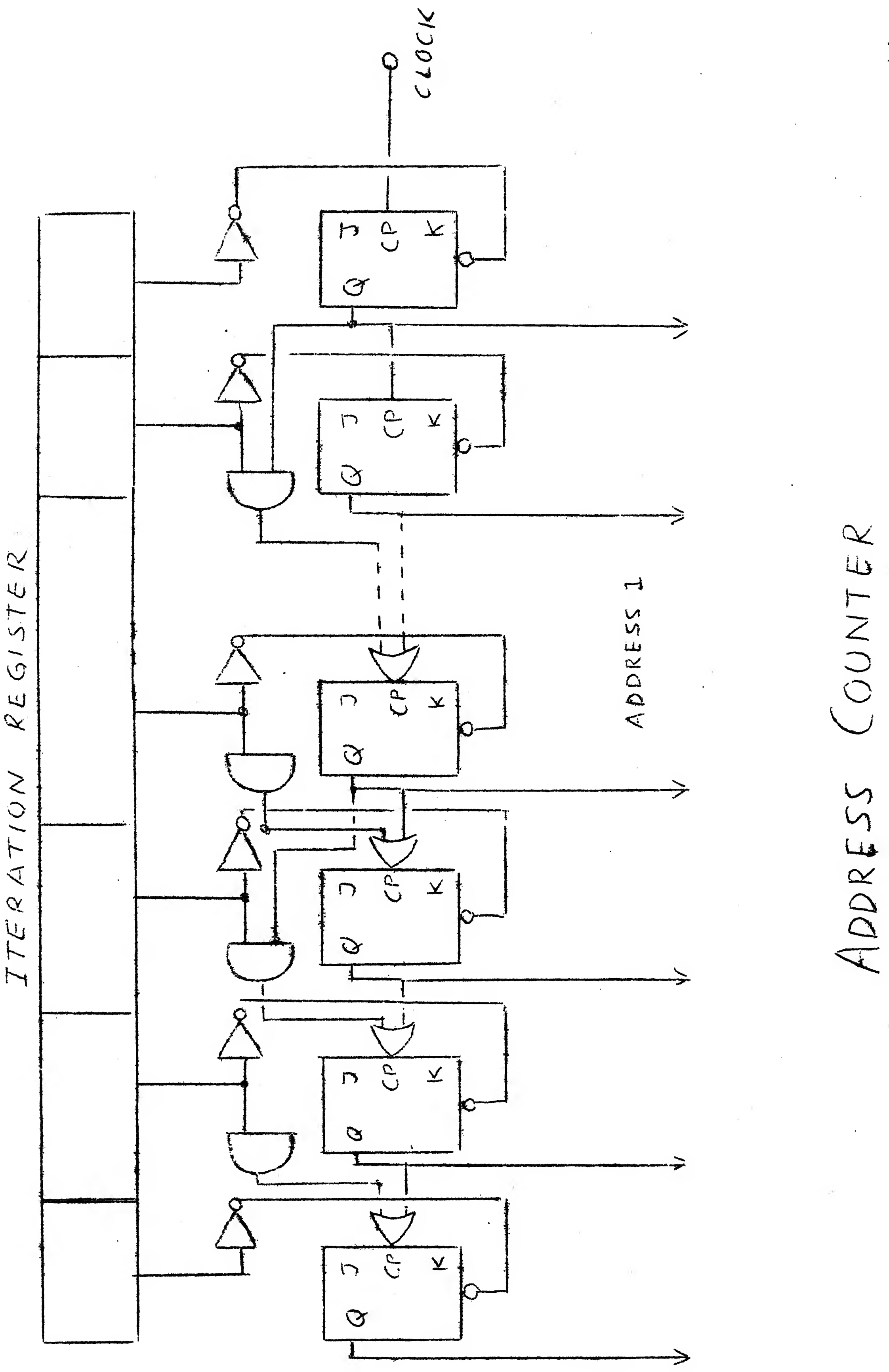


Figure 4.1

counter. Overflow of the computation counter signals the end of an iteration and causes the '1' in the iteration register to be shifted one place to the right. The output of computation counter is also used as input to some decoders to generate control commands as described in the next section.

4.5 Generation of 'R' and 'I' Commands

The control unit has to signal to the AU, when an extra complex multiplication is to be performed to generate a new weight. This also involves sending an increment pulse to the ROM counter and a load pulse to the ROM register. We will designate this ROM increment and load command as 'R' command. The other command is 'I' or 'interchange' command which signals the AU to interchange real and imaginary parts of the weight register and invert the sign bit of the real part. When both of these commands are absent, the AU performs only the butterfly calculation.

To generate the 'R' and 'I' command, the necessary logic can be determined by writing an example FFT flow chart for a small number of points say 32, making tables of the weights required in the different iterations, calculating the increments and when they are required, and detecting the logical patterns. These can be extrapolated for the higher point FFT. The final result for the 2K point FFT is given in Appendix A1. It turns out that particular bit patterns of computation counter must be detected which are different for different iterations. This can be easily done by simple decoders.

4.6 Other Control Commands

Apart from the above mentioned control signals, the control unit has to generate some other signals like Read Mrmory, Write Memory, load the registers, operate the one of four selector in the AU when storing back the processed data in the memory and so on. After the data has been loaded in registers, the control unit sends a trigger pulse to the AU control which starts the execution. Control unit then becomes idle and waits till the AU control signals end of computation. Then it starts storing the data in the memory, incrementing address counter, computation counter and detecting end of iteration.

4.7 Relative Speeds of AU and Main Control Unit

The execution time of the main control unit is much smaller compared to the AU execution time. However, there is a maximum limit on the clock frequency at which the control can operate. This is because of the 1 µsec. access time of the memory. The speed at which AU can operate is limited only by the Printed Circuit design, wiring delays and the speed of IC's. Hence the clock of AU can be much faster than the clock of the main control. This will reduce the mismatch between the speeds of the AU and the control.

4.8 Modularity

Since the signal flowgraph of FFT is highly symmetrical and repetitive, the control unit can be made to act for various number of points in radix 2 FFT. It can be easily made modular by a simple switch which causes the initial '1' to be loaded in different bits of the iteration register. For example, loading the '1' in the 11th bit from the right most bit will make the control unit operate for 2K number of points. In general, loading the initial 1 in k^{th} bit of the iteration register from the right most bit converts the control to work for 2^k number of points. Obviously the hardware must be initially designed for the highest number of samples to be processed.

REFERENCES

1. What is Fast Fourier Transform?
   By G-AE Subcommittee on Measurement Concepts.
   IEEE Transactions on Audio and Electroacoustics,
   Vol.AU15, pp.45-55, June 1967.
2. The time saver: FFT hardware.
   By R. Klahn, R.R. Shively, E.Gomez and M.J.Gilmartin,
   Electronics, pp. 92-97, June 24, 1968.
3. FFT Hardware Implementation, 'an overview'.
   By G.D. Bergland,
   IEEE Trans. on Audio and Electroacoustics,
   Vol.AU17, pp. 104-108, June 1969.
4. A short Bibliography on FFT
   By R.C. Singleton,
   IEEE Trans. on Audio and Electroacoustics,
   Vol.AU17, pp. 166-169, 1969.

Appendix A1

GENERATION OF 'R' AND 'I' COMMANDS

'I' command causes the contents of 'g' and 'h' registers to be interchanged. This is not generated in the first iteration since only one weight W^o is required. In the second iteration it is generated once, after half the computations (N/4) are over. This corresponds to the most significant bit of the computation counter being 1 and all the remaining less significant bits, (LSB's) being 0. In the third iteration, 'I' command is generated twice, once when N/8 computations are over, and then again after 3N/8 computations are over. In this case the second most significant bit of computation counter is 1 and LSB's are 0. The most significant bit may be either 0 or 1, i.e., in a 'don't care' condition. In general, in the k^{th} iteration 'I' command is generated $2^k/4$ times, the first 'I' command being given after $N/2^k$ computations are over. This corresponds to the state in which the (k-1)th most significant bit of the computation counter is 1 and the LSB's are 0. All the higher significant bits may be in 'don't care' condition. A simple decoder, with the Iteration Register outputs and the computation counter outputs as its inputs, can generate this command.

'R' command increments the ROM counter and loads the new incremental coefficient in the ROM buffer register. This command is not generated in the first two iteration since no new weight is required. It is generated once in the third iteration, thrice in the fourth iteration and seven times in the fifth iteration. In general, in the k^{th} iteration, it is generated $(2^{k-2}-1)$ times. The occasion when it is generated for the first time corresponds to the state of the computation counter in which the (k-2)th most significant bit is 1 and the LSB's 0. The more significant bits may be in 'don't care' condition. Thereafter the 'R' command is generated after every $N/2^{k-1}$ computations. This too can be synthesized using a decoder similar to the one for 'I' command generation.

Appendix A2

LIST OF CONTROL COMMANDS

(1) Master clock.

(2) Trigger pulse (AU enable).

(3) AU complete (to main control from AU).

(4) 'R' command.

(5) 'I' command.

(6) ROM counter increment pulse.

(7) ROM coefficient load pulse.

(8) ROM buffer register mode.

(9) Mode 'c'.

(10) Mode 'd'.

(11) Mode 'e'.

(12) Mode 'f'.

(13) Load 'c'.

(14) Load 'd'.

(15) Load 'e'.

(16) Load 'f'.

(17) Select 'c'.

(18) Select 'd'.

(19) Select 'e'.

(20) Select 'f'.

(21) ROM counter clear pulse.

(22) Initialize 'g,h' (at the start of each iteration).

(23) Toggle pulse to sign 'g' flag (with the I command).

Master clock is fed continuously to the AU. Trigger pulse enables the AU. After execution, AU control sends 'AU complete' signal to the main control. 'R' and 'I' commands have been explained previously. ROM counter increment pulse increments the ROM counter. ROM coefficient load pulse loads the new incremental coefficient in the ROM buffer register. Mode commands are used while loading the data from the memory into the data registers. Load pulses write the data in the registers selectively. Select commands selects one of the four real numbers to be stored back in the memory one by one. ROM counter clear pulse and 'Initialize g,h' pulse are given at the start of every iteration. Simultaneously with I command, a toggle pulse is given to the sign 'g' flag.

Appendix A3

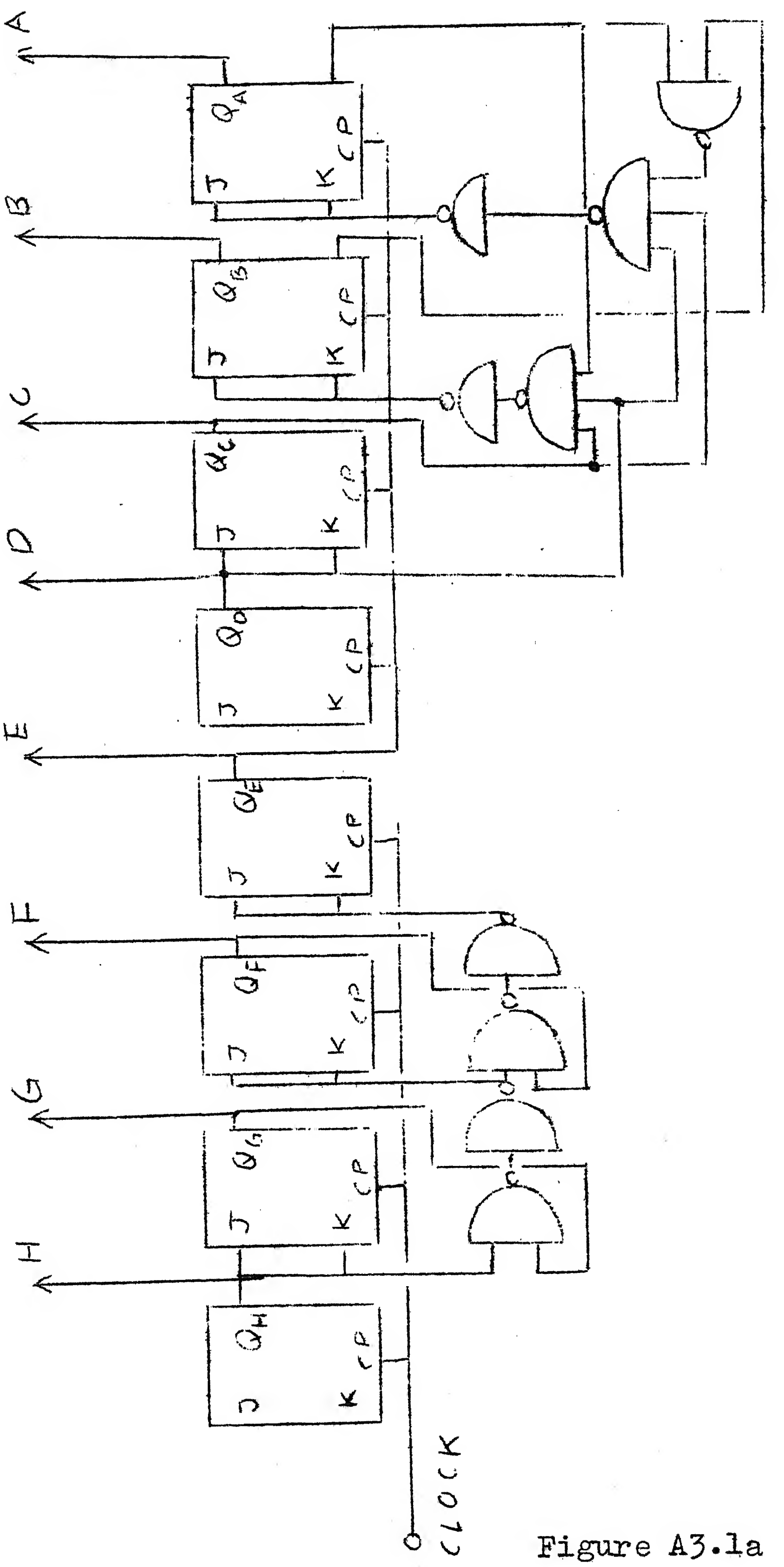
FABRICATION

The complete arithmetic unit is housed in a single cabinet containing two racks. Each rack has the capacity to hold sixteen printed circuit cards. The upper rack contains data registers, diode matrices constituting the ROM, ROM counter, ROM buffer register and 'one of four' selector. The lower rack contains the processing hardware i.e. multipliers, adder, subtractor and associated gating and flags. It also contains the cards constituting the AU control. The total number of cards is 30, including connector cards which are used to bring connections from back side to front side and viceversa.

There are only two panel switches, one for power and the other for clearing various sequential circuits at the start of the experiment. Two parallel data buses, each with 15 lines, connects the AU to the main memory. One data bus is for input and one for output. One 23 line bus carries the various control commands to and from the main control unit.

The detailed circuit diagrams are given on the following pages. Circuit number A3.1a and A3.1b are on the same card. Rest of the circuits correspond to a single card each.

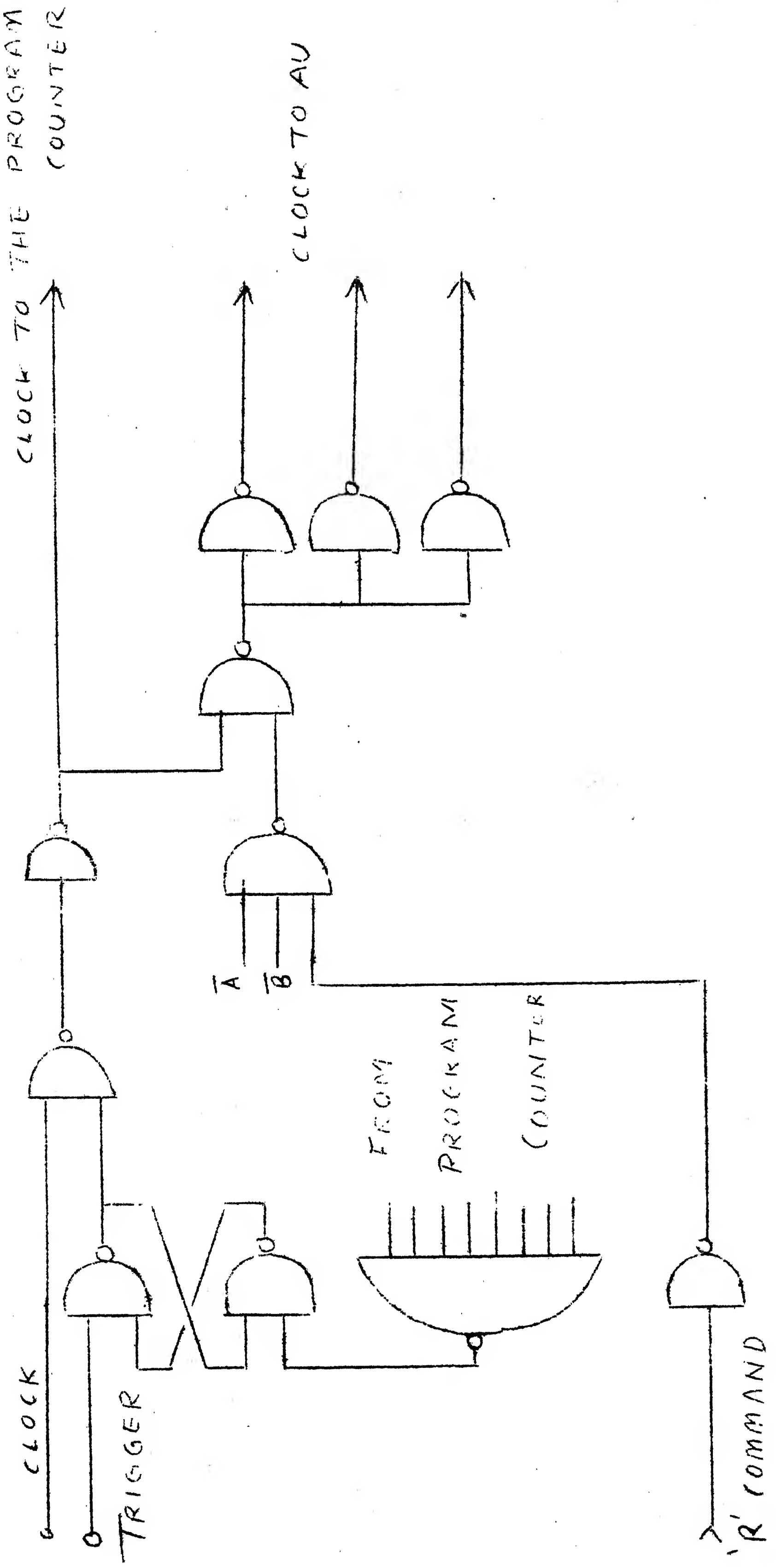
PC NO. 88/75



PROGRAM COUNTER
12 x 16 STATE

Figure A3.1a

PC NO. 88/75



CYCLE CLOCK GENERATOR FOR THE AU

Figure A3.1b

PC NO. 87/75

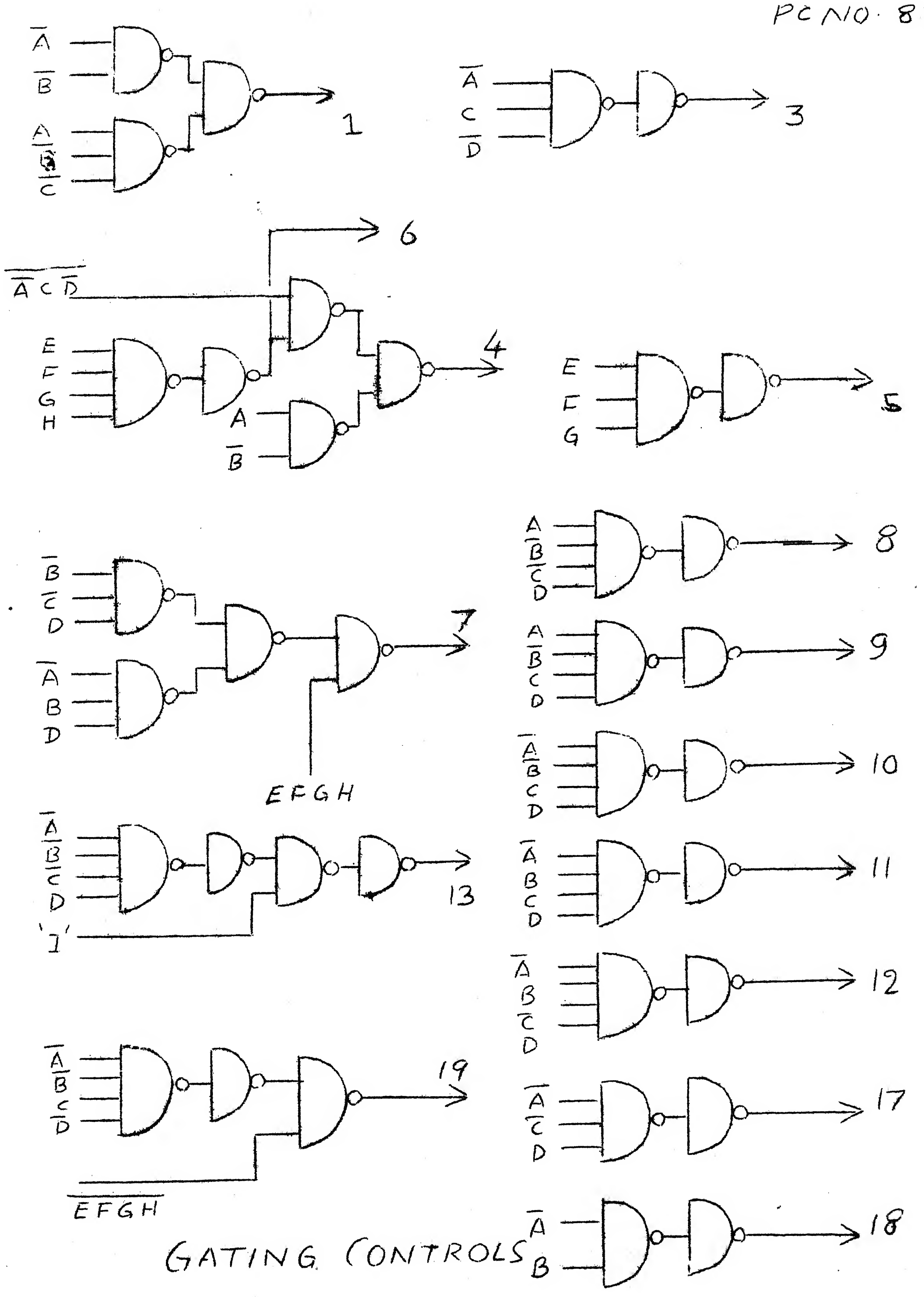
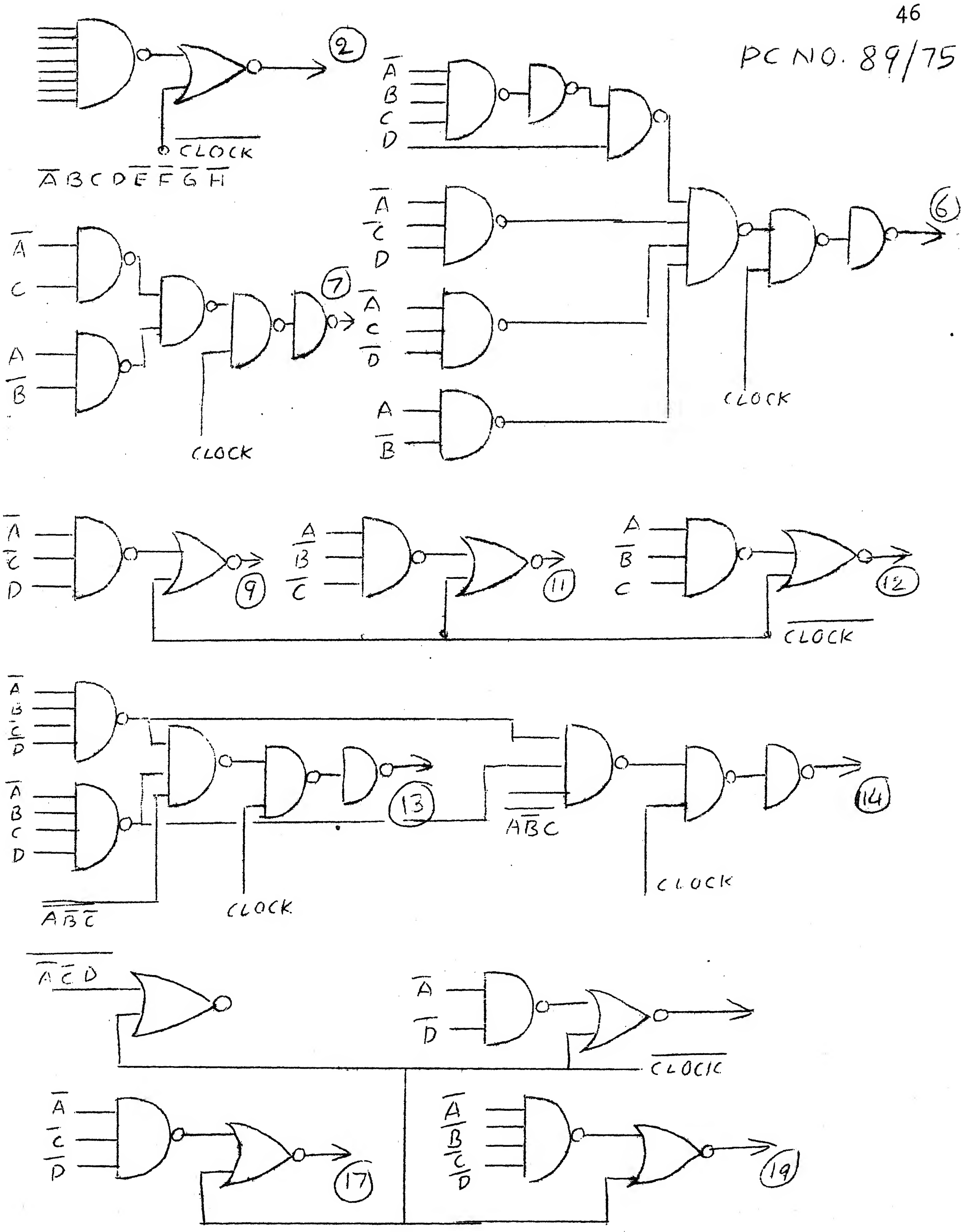


Figure A3.2

PC NO. 89/75

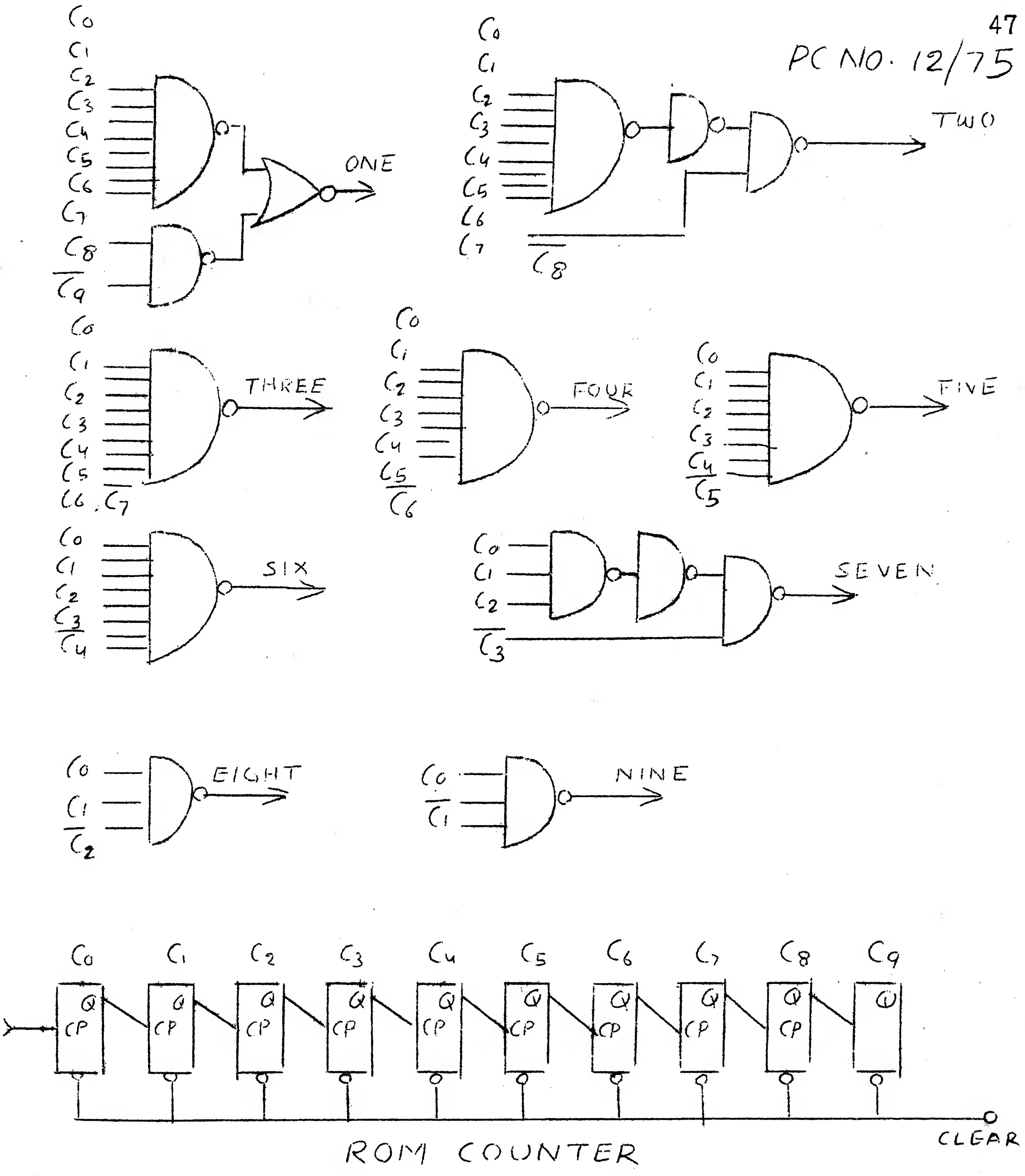


$\overline{A}BCD\overline{E}\overline{F}\overline{G}\overline{H}$
$\overline{CLOCK}$
2
$\overline{A}$ B C D
$\overline{A}$ $\overline{C}$ D
$\overline{A}$ C $\overline{D}$
A $\overline{B}$
6
CLOCK
$\overline{A}$ C
A $\overline{B}$
7
CLOCK
$\overline{A}$ $\overline{C}$ D
9
A $\overline{B}$ $\overline{C}$
11
A B C
12
$\overline{CLOCK}$
$\overline{A}$ B $\overline{C}$ $\overline{D}$
$\overline{A}$ B C D
$\overline{A\overline{B}\overline{C}}$
CLOCK
13
$\overline{A\overline{B}C}$
CLOCK
14
$\overline{\overline{A}\overline{C}D}$
$\overline{A}$ $\overline{D}$
$\overline{CLOCK}$
$\overline{A}$ $\overline{C}$ $\overline{D}$
17
$\overline{A}$ $\overline{B}$ $\overline{C}$ $\overline{D}$
19

CLOCK GATING CONTROLS

Figure A3.3

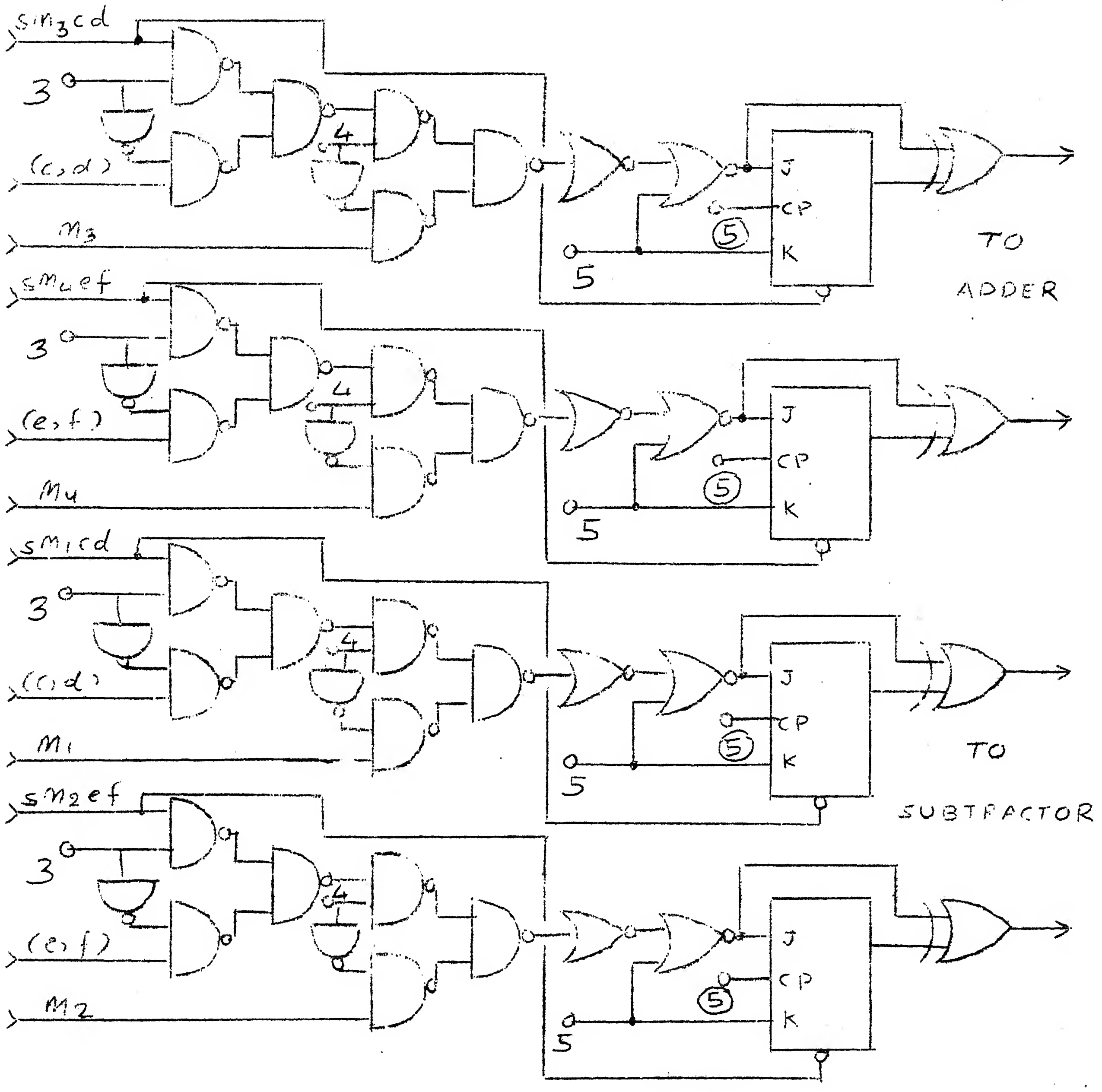
PC NO. 12/75



C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 $\overline{C_9}$
ONE
C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7 $\overline{C_8}$
TWO
C_0 C_1 C_2 C_3 C_4 C_5 C_6, $\overline{C_7}$
THREE
C_0 C_1 C_2 C_3 C_4 C_5 $\overline{C_6}$
FOUR
C_0 C_1 C_2 C_3 C_4 $\overline{C_5}$
FIVE
C_0 C_1 C_2 C_3 $\overline{C_4}$
SIX
C_0 C_1 C_2 $\overline{C_3}$
SEVEN
C_0 C_1 $\overline{C_2}$
EIGHT
C_0 $\overline{C_1}$
NINE
C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9
Q
CP
CLEAR
ROM COUNTER

ROM DECODER

Figure A3.4

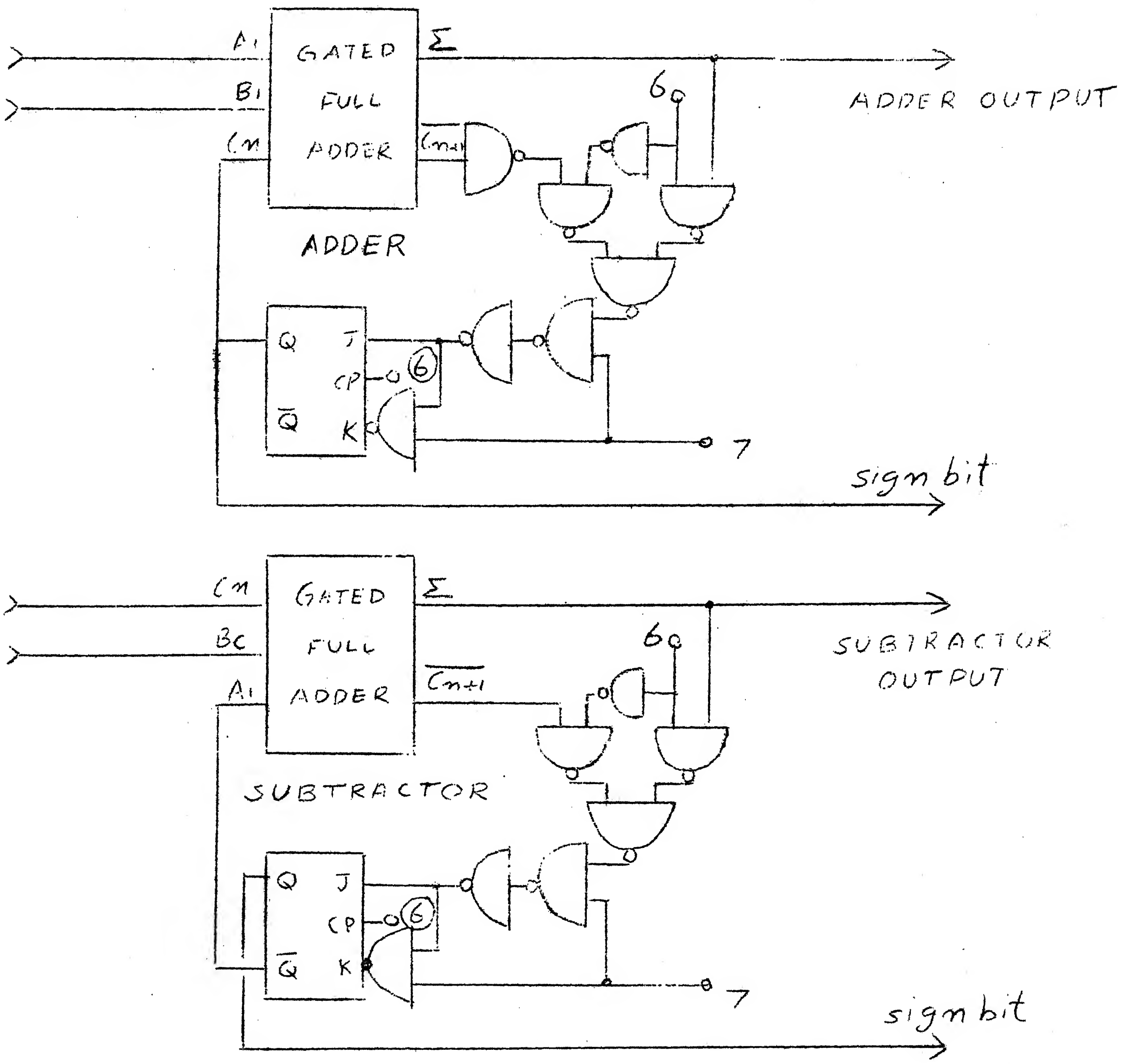


PC NO. 16/75

ADDER/SUBTRACTOR 'A'

Figure A3.5

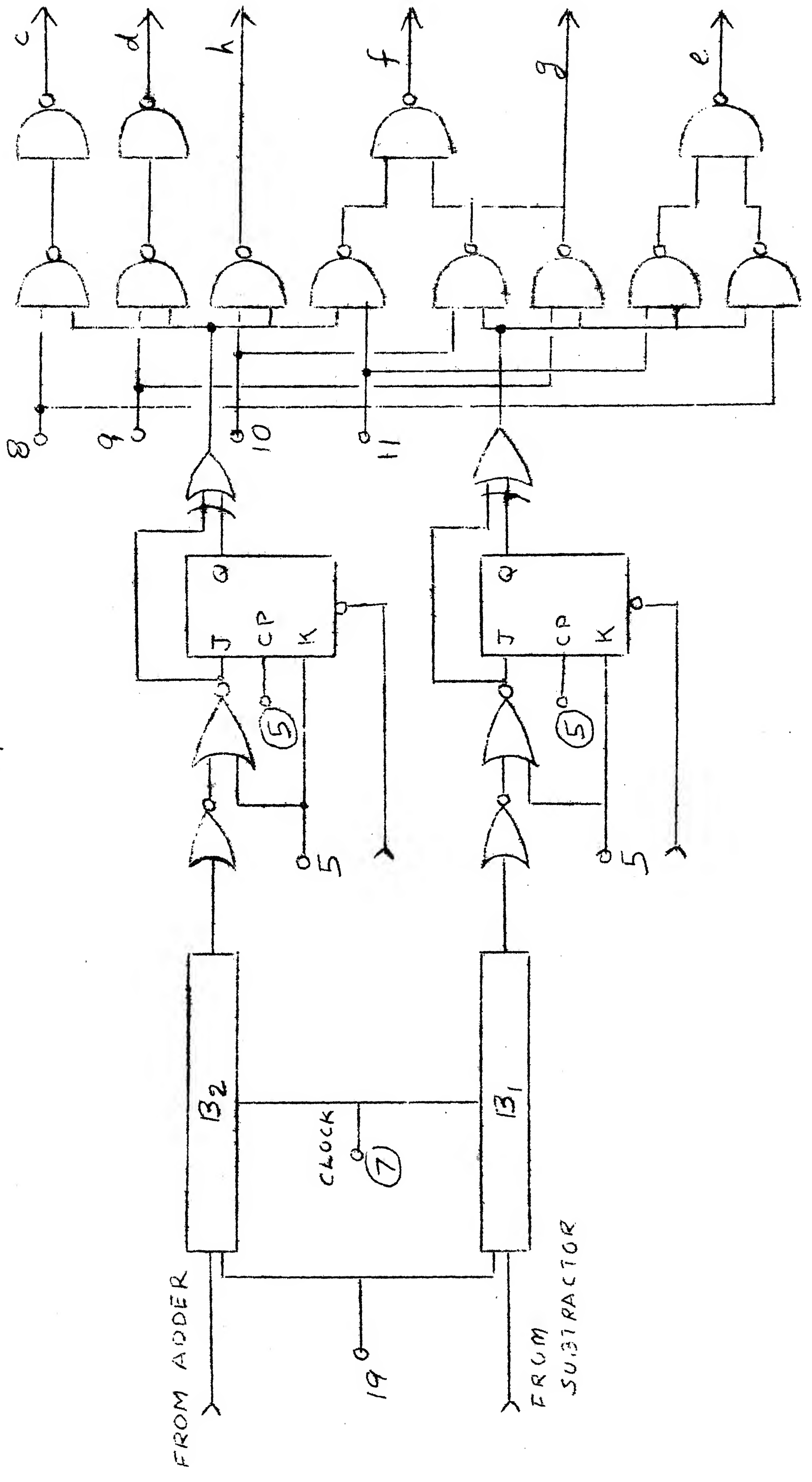
PC NO 23/75



ADDER/SUBTRACTOR 'B'

Figure A3.6

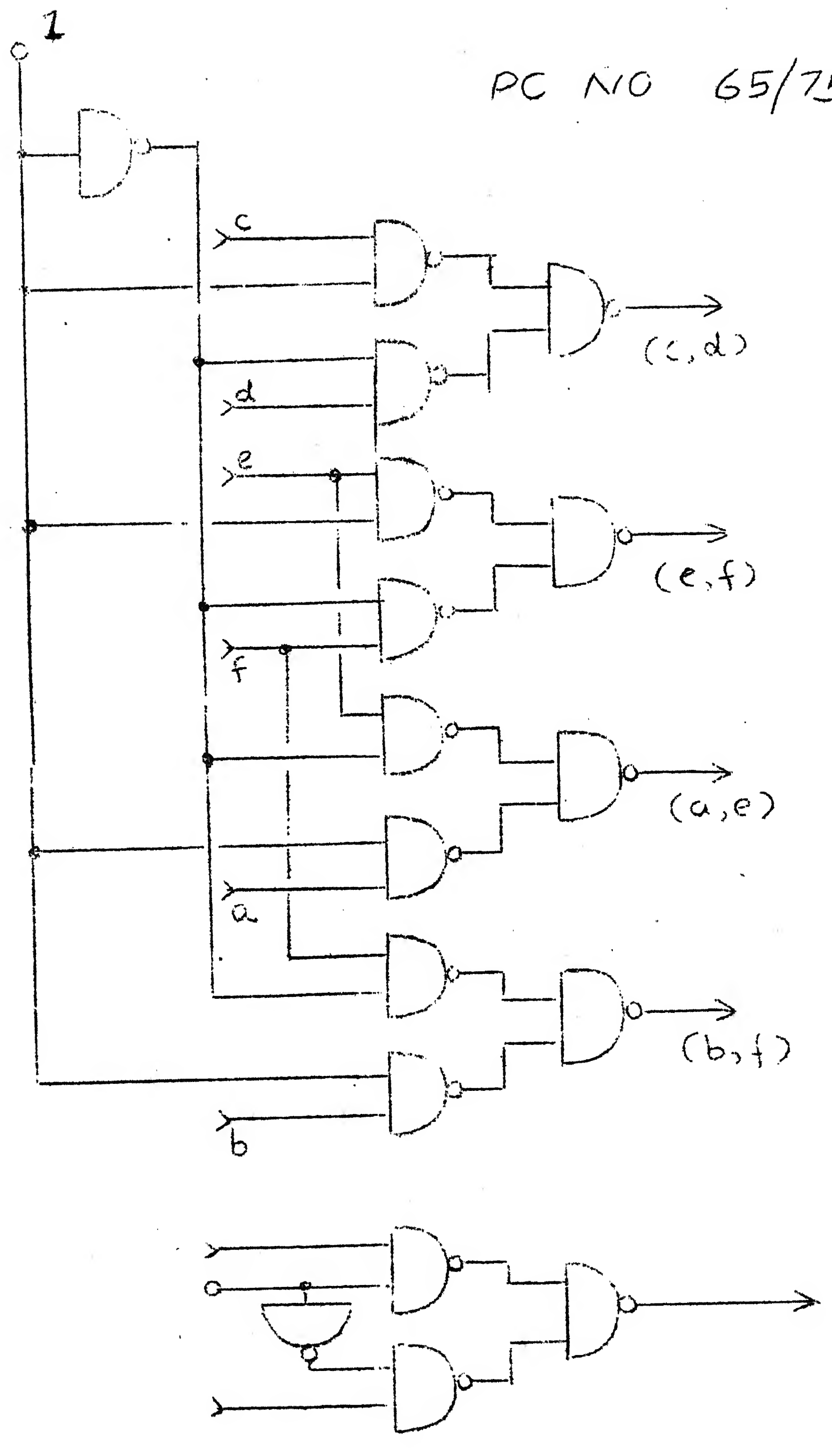
PC NO 28/75



ADDER/SUBTRACTOR 'C'

Figure A3.7

PC NO 65/75



GATING FOR DATA SELECTION

Figure A3.8

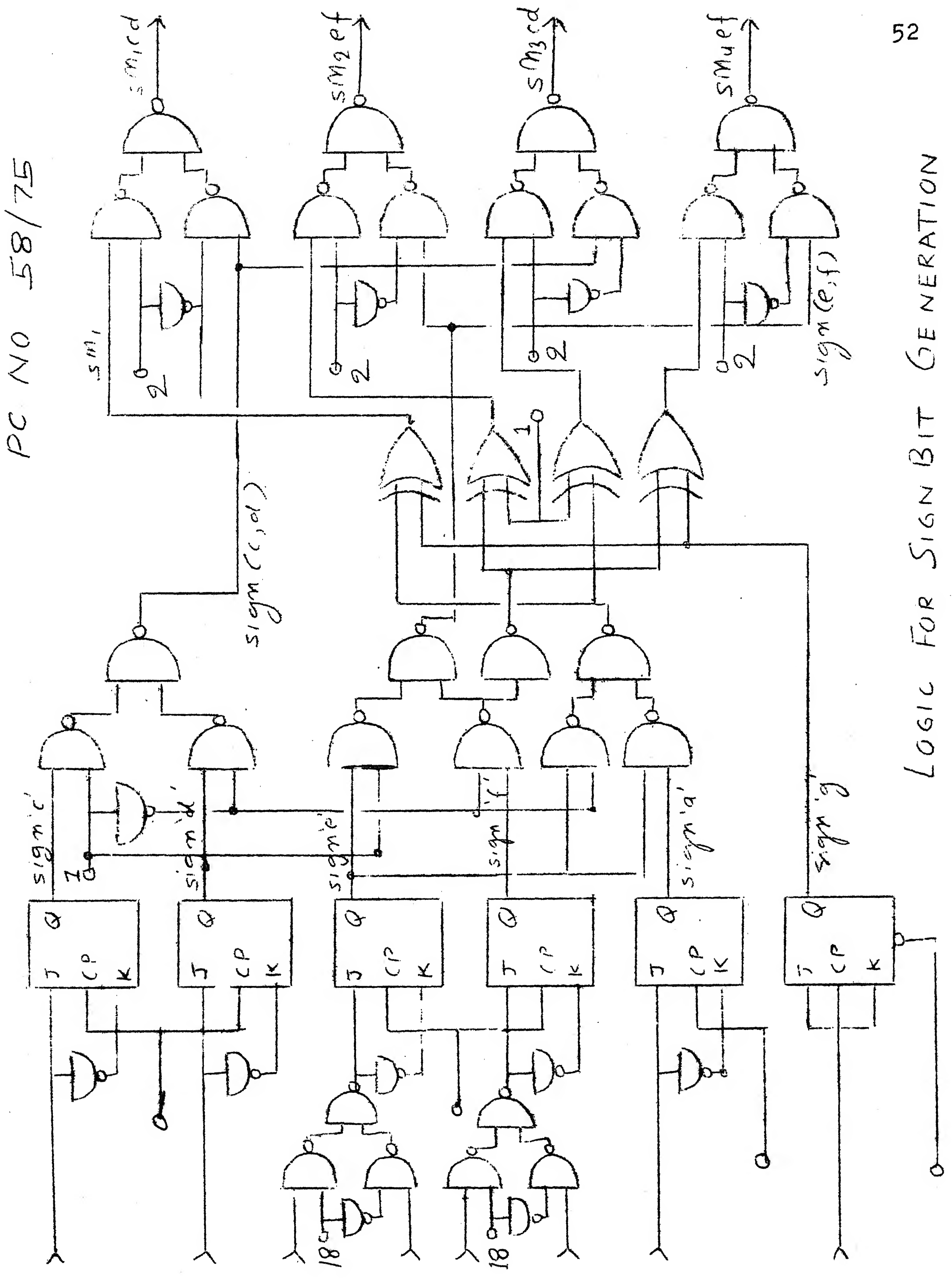
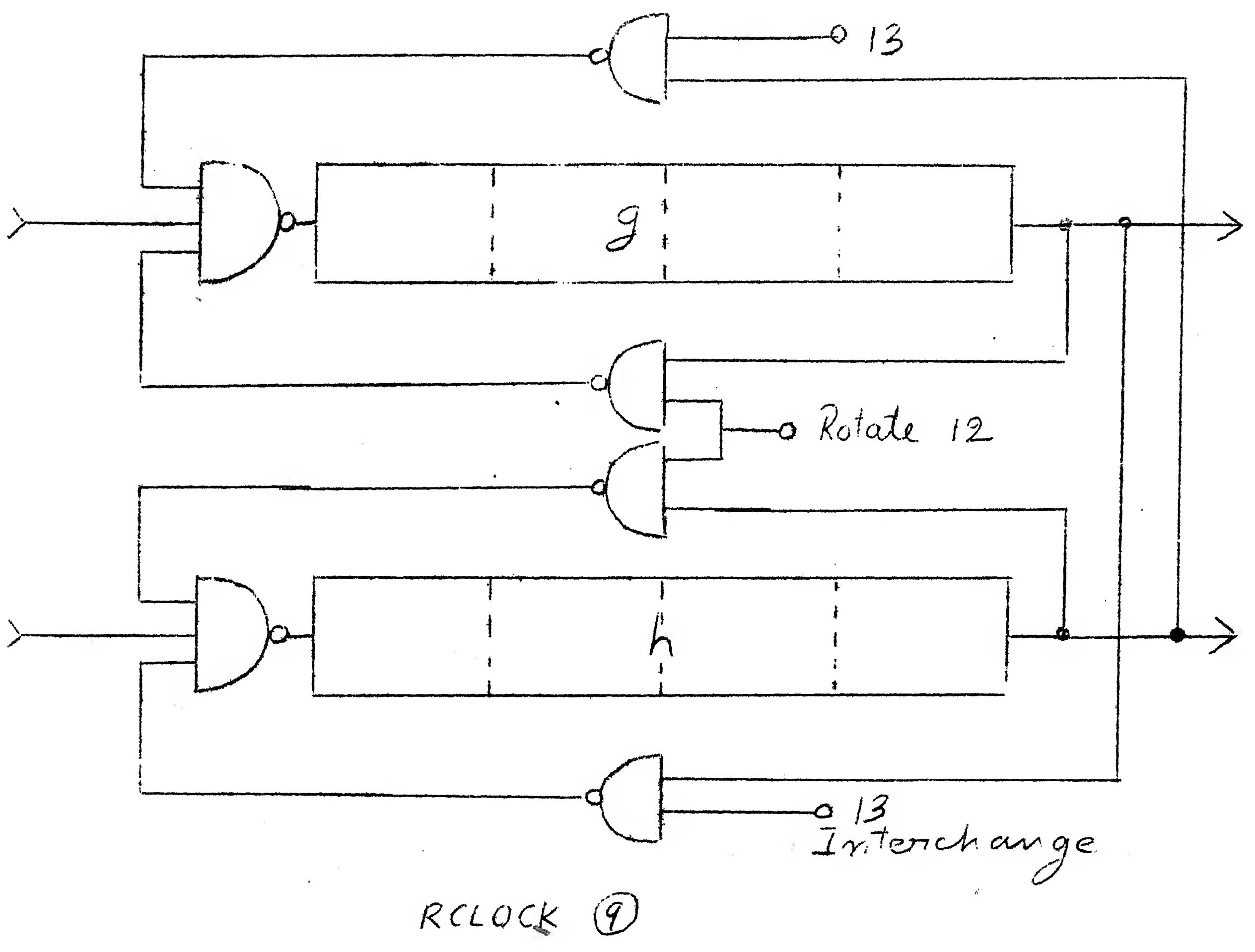

PC NO 58/75
LOGIC FOR SIGN BIT GENERATION
sm1cd
sm2ef
sm3cd
sm4ef
sm1
sign(c,d)
sign(e,f)
sign'c'
sign'd'
sign'e'
sign'f'
sign'a'
sign'g'
Q
J
CP
K
1
2
18

Figure A3.9

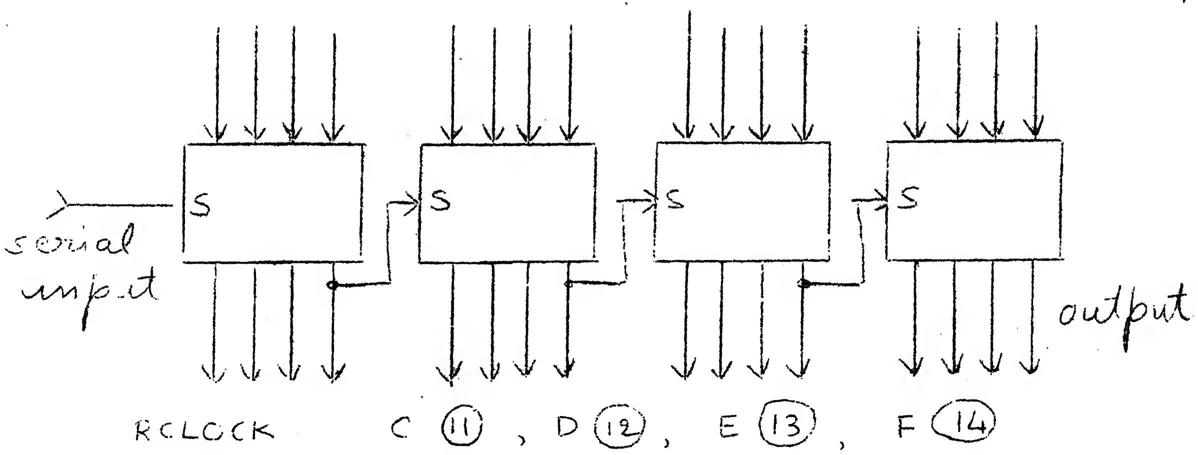
PC NO 27/75



TRIGONOMETRIC COEFFICIENT REGISTER

Figure A3.10

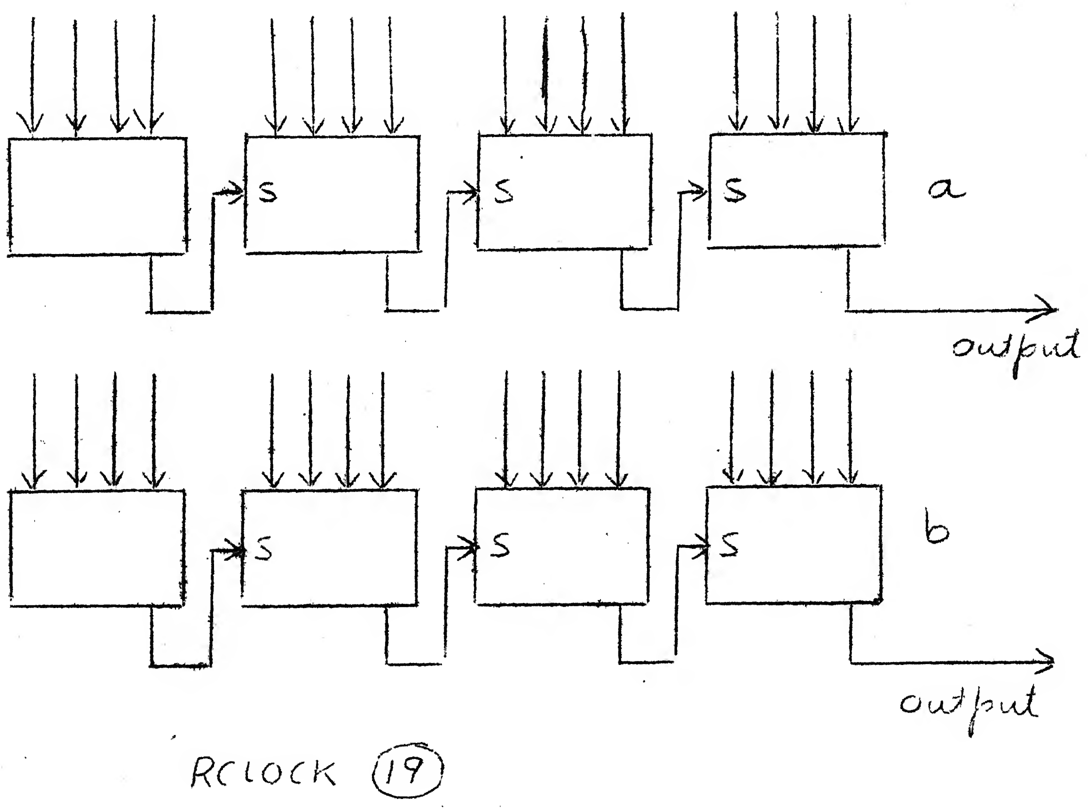
PC NO 38/75



DATA REGISTER (FOUR)

Figure A3.11

PC NO 39/75



ROM BUFFER REGISTER

Figure A3.12

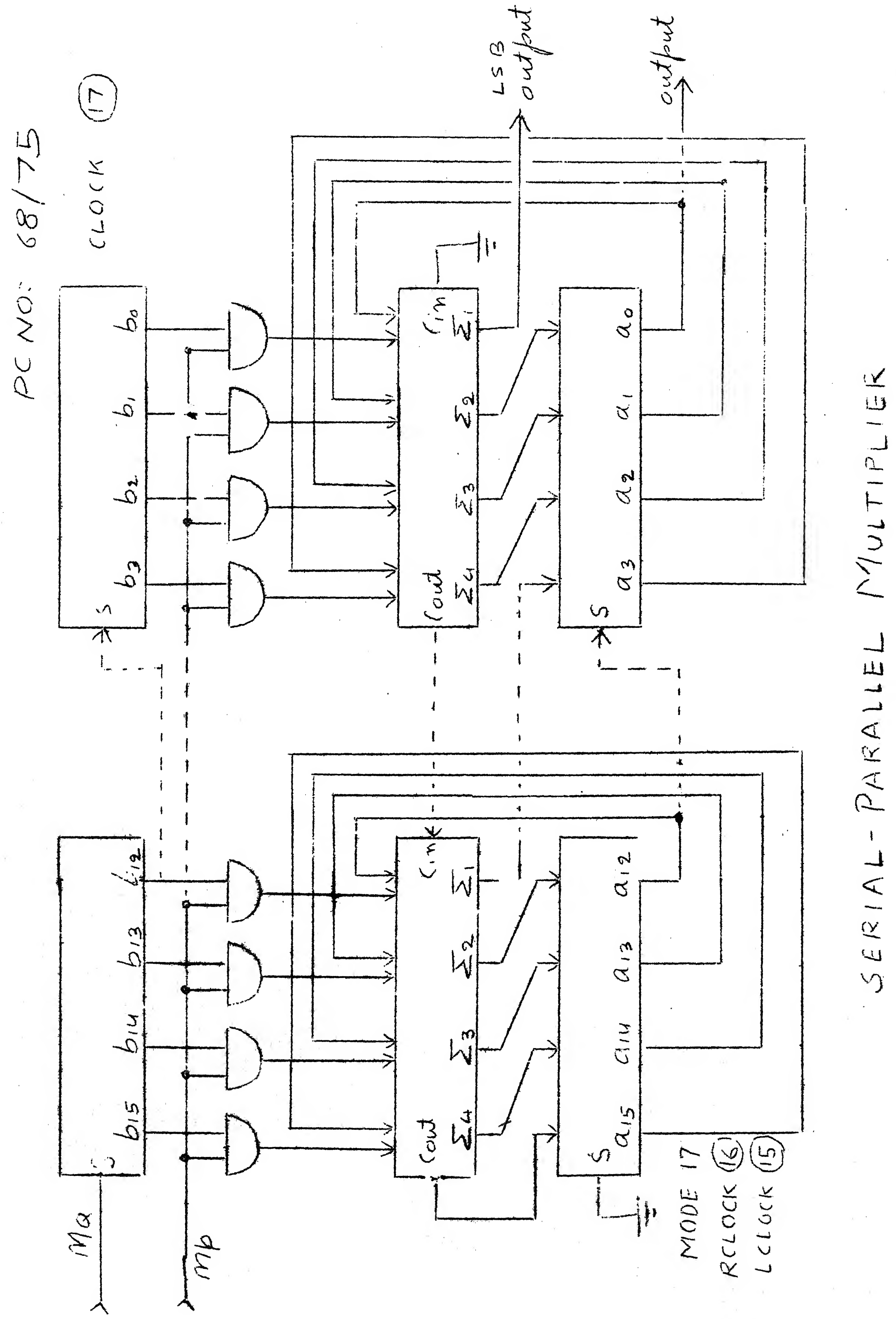
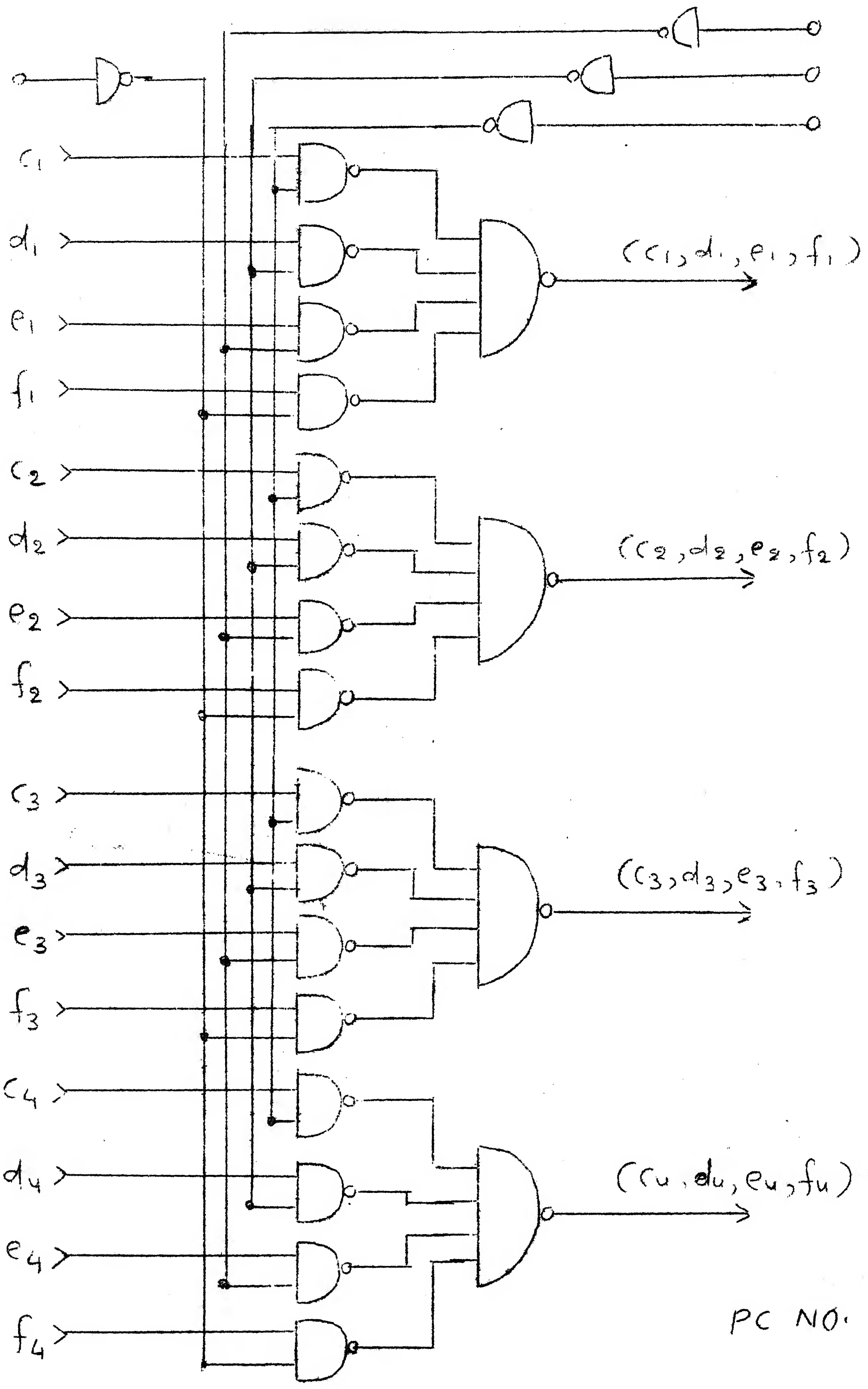

PC NO: 68/75
CLOCK (17)
SERIAL-PARALLEL MULTIPLIER
LSB output
output
b_0 b_1 b_2 b_3
b_{12} b_{13} b_{14} b_{15}
C_{in} Σ_1 Σ_2 Σ_3 Σ_4 C_{out}
a_0 a_1 a_2 a_3
a_{12} a_{13} a_{14} a_{15}
S
Ma
Mb
MODE 17
RCLOCK (16)
LCLOCK (15)

Figure A3.13



PC NO. 92/75

ONE OF FOUR SELECTOR

Figure A3.14.

EE-1975-M-KAN-FAS